FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Application usage Profiling and Forecasting in Shared Cloud Systems

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering & Internet Computing

eingereicht von

Péter Patonai
Matrikelnummer 1027183

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung: Priv.-Doz. Dr. Ivona Brandić

Wien, 01.08.2014 ____________________ ____________________
(Unterschrift Verfasser) (Unterschrift Betreuung)

Application usage Profiling and Forecasting in Shared Cloud Systems

MASTER'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Software Engineering & Internet Computing

by

Péter Patonai
Registration Number 1027183

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Priv.-Doz. Dr. Ivona Brandić

Vienna, 01.08.2014

(Signature of Author) (Signature of Advisor)

Erklärung zur Verfassung der Arbeit

Péter Patonai
Margaretenguertel 76-80/11/15, 1050 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen -, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

(Ort, Datum) (Unterschrift Verfasser)

Acknowledgements

This thesis would not have been possible without the support of many people.
I want to express my gratitude to my supervisor Dr. Ivona Brandić who was abundantly helpful and offered invaluable assistance, support and guidance.

I am especially grateful for my family who has been always supportive of me. In loving memory of my father who taught me the importance of knowledge. To my mother for her constant and unconditional love. And to my sister for her encouragement.

A special thanks goes to my girlfriend Michele, who had the hardest part of the work, namely dealing with my complaining and agony throughout the completion.

Abstract

This thesis addresses the problem of elastic cloud resource scaling. Cloud providers have to offer on-demand resource provision for time varying workloads. This often leads to SLA violations on application level, or energy waste due to providing more physical resources for the hosted applications than needed. Providing always as much physical resources as needed, results in QoS stability for the hosted applications. Also, significant reduction of electricity consumption can be achieved by dynamically turning down unused resources. This makes the cloud providers more competitive.

Thereby we describe an adaptive and scalable scheduling infrastructure. This monitors the cloud applications, selects automatically an appropriate statistical forecasting method. Then, based on the forecast values, optimizes the resource allocation in the cloud. The contribution of this thesis is three-fold: (1) It presents a detailed survey about the available forecasting and time series analysis tools. (2) It provides a flexible and automatic classification framework to choose the best fitting forecasting method, based on the features of the monitored traces, such as trend, seasonality, and variance. (3) In the course of the work, a cloud simulator is developed in order to compare different forecasting methods, how accurately they can predict SLA violations.

During the work it was observed that a high proportion of cloud applications are exhibiting daily periodical fluctuations in terms of resource needs. Thus, we are focusing on seasonal forecasting, by applying the Fourier transformation based forecasting, the Holt-Winters exponential smoothing, the neural network autoregression, and the STL decomposition based forecasting. These seasonal forecasting tools are evaluated by using the implemented cloud simulator. For the test we use synthetically generated traces of applications with periodically fluctuating cpu, memory, network, and disk I/O usage. We compare the used forecasting methods by the number of correct, incorrect and false positive predictions of SLA violations. In relation to the simulation we discuss additional aspects, such as heuristic based target machine selection and migration, the usage of forecast values to create scheduling rules, and the adaptive error corrections. The ultimate aim of the simulator is to test the interaction of various approaches and the effect of those approaches on cloud utilization and on violation detection.

Kurzfassung

Diese Arbeit beschäftigt sich mit dem Problem des elastischen Skalierens von Cloud Resourcen. Cloud Anbieter stellen bedarfsorientiert Ressourcen für Cloud Anwendungen bereit, deren Ressourcennutzung sich mit der Zeit variiert. Dies führt oft zu SLA Verletzungen auf der Applikationsebene, oder zum höheren Stromverbrauch durch das Bereitstellen von mehr physikalischen Ressourcen für die Cloud Anwendungen, als notwendig. Eine optimale Zuteilung von physikalischen Ressourcen ergibt einerseits QoS Stabilität. Anderseits, durch das Abschalten von ungenutzten Hardwareressourcen kann der Energieverbrauch in der Cloud signifikant reduziert werden, wodurch der Cloud Anbieter wettbewerbsfähiger wird.

Wir beschreiben eine anpassungsfähige und skalierbare Infrastruktur für Cloud Anbieter. Diese ist in der Lage, Cloud Anwendungen zu überwachen, automatisch eine angemessene statistische Voraussagemethode zu wählen und basierend auf den vorausgesagten Daten die zugewiesenen Ressourcen in der Cloud zu optimieren. Der Beitrag dieser Arbeit besteht aus drei Teilen: (1) Es ist eine Studie durchgeführt worden, in der wir die verfügbaren vorhersage Werkzeuge und Methoden zur Zeitreihenanalyse identifiziert haben. (2) Wir haben ebenfalls ein flexibles und automatisches Klassifizierungsframework entwickelt, mit dem es möglich ist die beste Vorhersagemethode, basierend auf dem Trend, der Tageszeit und der statistischen Varianz zu identifizieren und auszuwählen. (3) In Rahmen dieser Arbeit wurde ein Cloud Simulator entwickelt, der es ermöglicht die verschiedenen Vorhersagemethoden zu vergleichen. Dies geschieht insbesondere mit Hinblick auf die Genauigkeit mit der SLA Verletzungen vorhergesagt werden können.

Durch die Studie konnten wir einen hohen Anteil von tageszeitenabhängigen Cloud Anwendungen identifizieren. Aus diesem Grund beschäftigen wir uns in dieser Arbeit mit tageszeitabhängigen Vorhersagemethoden auf Basis der Fouriertransformation, Holt-Winter exponentieller Glättung und automatischer Regressionsanalyse in neuronalen Netzwerken, sowie STL Zerlegung. Mit der Hilfe der Cloud Simulator sind diese Vorhersagemethoden ausgewertet. Für die Simulation verwenden wir synthetische Lastdaten der Anwendungen mit wechselnden Werten für CPU, Speicher, Netzwerkbandbreiten und Festplatten I/O. Die Auswertung erfolgt durch Vergleich der Anzahl von richtig, falsch und falsch positiv identifizierten SLA Verletzungen. Im Verhältnis zu der Simulation werden außerdem weitere Aspekte wie Heuristik basierte Anwendungsmigration betrachtet, bei der die vorhergesagten Daten als Basis für Schedulingregeln dienen. Es kommt zu dem eine adaptive Fehlerkorrektur zum Einsatz. Das Ziel des Simulators ist es letztendlich die Interaktion von verschiedenen Ansätzen zu testen, sowie deren Auswirkungen auf die Cloud Auslastung und auf das Erkennen von SLA Verletzungen.

Contents

List of Figures

List of Tables

Abbreviations

ACF	Autocorrelation Function
AIC	Akaike information criterion
AR	Autoregression
ARIMA	Autoregressive integrated moving average
DSHW	Double Seasonal Holt-Winters Exponential Smoothing
ETS	Exponential Smoothing
FFT	Fast Fourier Transformation
HW	Holt Winters Exponential Smoothing
MA	Moving Average
ME	Mean Forecast Error
NNAR	Neural Network Autoregression
PAA	Piecewise Aggregate Approximation
PACF	Partial Autocorrelation Function
PM	Physical Machine
PLA	Piecewise Linear Approximation
SAX	Symbolic Aggregate approximation
SLA	Service Level Agreement
STL	Seasonal and Trend decomposition using Loess
TSDB	Time Series Database
VM	Virtual Machine
QoS	Quality of Service

CHAPTER 1

Introduction

1.1 Motivation

Data-centers worldwide have a significant part in the overall power consumption. IT-systems consumes 10.5% of energy in Germany (Deutscher Bundestag,2010) and they produce 2% of worldwide CO_2 emissions [58]. Energy consumption has been growing in the recent years and the growth is expected to continue. Many studies show that there is a huge unused potential to cut power consumption with improving the utilization of existing resources. Utilization levels in some data centers is as low as 15%. Figure 1.1 below shows the average CPU utilization of more than 5000 servers during a six-month period. As a result it can be seen that servers are seldom operating on near maximum utilization and the most of the times they are operating between 10 and 50 percent of their maximum utilization levels [75] [8].

The low utilization of data centers basically means that providers are burning money: they are wasting electricity with running idle machines, while they could fit more running applications on the platforms increasing their profit.

Consolidating server workload in cloud computing environment is an effective way to achieve better utilization and so reduce power consumption. The aim of consolidation is to concentrate the workloads on minimal number of physical servers. Shutting down the idle resources in the cloud is a big energy saver since a running server consumes upwards of 60%-70% of its maximum power consumption, even if it does not carry load [75] [10].

A cloud environment usually consist of a large number of physical machines on that one can host an even larger number of applications (workloads) using virtualization techniques. Virtualization is a key technique in the process that allows to dynamically assign and release resources to and from hosted applications, or migrate application on different machine if the resource demand is outgrowing the physical machine. The Cloud scheduler is the entity that controls the scheduling of the virtualized workloads on the physical resources. Commonly the schedulers work on a reactive way trying to adjust the provisioned resources to the demands using threshold based rules or other heuristical algorithms.

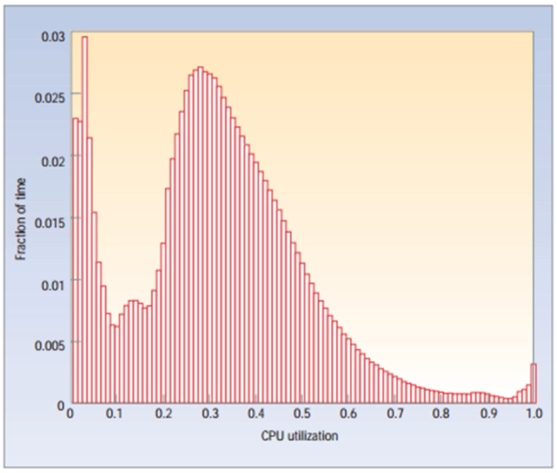


Figure 1.1: Utilization Level of 5000 randomly selected google servers over a 6 Month period [8]

This master thesis describes a mechanism to optimize the scheduling process. Our approach is to monitor the resource needs of the applications running in the cloud and based on historical usage patterns we predict how the resource demand will follow in the future and make proactive scheduling decisions. The challenge is to minimize power consumption in the cloud while providing enough resources to all application to prevent Service Level Agreement (SLA) violations, and while doing this expose as little as possible additional cost to the cloud.

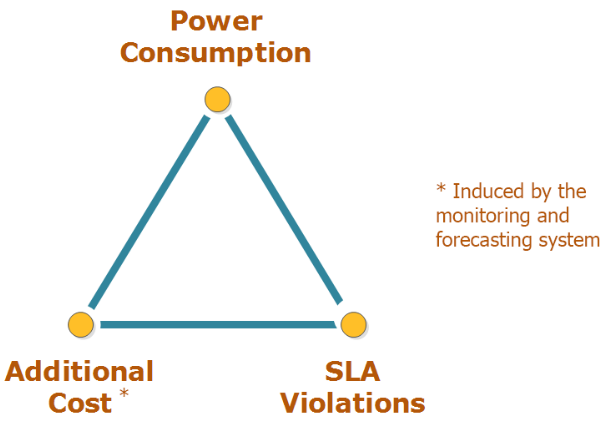


Figure 1.2: The three optimization objectives are mutually exclusive. Optimization of all three at once is not possible, but keeping the balance between them is important. The balancing criteria is that the additional cost induced by the monitoring and forecasting infrastructure should be less than the cost spared by having less SLA violations and idle cloud resources.

These requirements are mutually exclusive: all three can not be minimized at once. For example minimal number of SLA violations can be easily achieved with just hosting each application on a different machine, but that also means that the power consumption is not optimal. We want to achieve a balance between these three targets so that we expose some overhead in the cloud and in return we minimize the number of SLA violations, improve fairness of resource assignments, and improve utilization thus minimize the power consumption of the cloud. Based on the observation that many (web)applications do have similar number of user accesses in the same time of the day, we are concentrating on predicting future resource usage of applications with seasonal cyclical usage patterns on a long term. To prove the validity of the idea we implemented a simulation that runs several applications with changing resource needs on as many machines as needed and that adapts to the changes of the applications behavior in a reactional manner. We extend this simulation by providing different forecasting mechanisms on the resource needs of the applications and we measure how many times SLA violation could have been prevented based on the forecast values.

1.2 Problem Statement

Cloud computing has the advantage of dynamically scaling cloud resources as they are needed. This includes turning on and down physical hardware resources. This scaling process is done by each cloud provider differently. Amazon for example uses reactive resource scaling, which means if an application „outgrows“ the physical hardware boundaries it simply adds additional hardware resources. This has the weakness that the reaction takes time (detecting the underprovision and turning on physical hardware) during that significant quality of service degradation can arise which leads to penalty payment for the cloud provider (if service level agreement between provider and customer is agreed upon).

Other cloud providers simply overprovide applications and keep a lot already turned on backup physical resources to be used as soon as needed. In this case no QoS degradation and SLA violation arises but instead of penalty payment the cloud provider is facing with huge electricity waste.

Finding the line between this two approaches to react on violations before happening without leaving unused resources turned on would ensure significant economical advantage for cloud providers.

1.3 Aim Of The Thesis

The aim of this thesis is to devise a general cloud scheduling architecture that optimizes the resource allocation process so that SLA violations can be detected before they arise and be handled while keeping just as much physical resources in the cloud turned on as needed. This is done by monitoring the cloud applications resource usage changes over time and based on the inspected historical traces, use different models and methods to forecast the possible violations so that the cloud can adapt to future states in time. This way cloud provider saves electricity while maintaining overall fairness among the hosted applications.

1.4 Methodological Approach

The thesis starts by reviewing the related technologies, concepts and research in the topic of cloud scheduling, energy efficiency in the cloud, SLA violation detection and reaction methods. We first provide the theoretical architecture of scalable and adaptive application profiling system, that monitors the application resource usage traces, selects automatically the best possible forecast method based on the trace features, and that uses forecasting to make cloud scheduling decisions. This will result in better overall cloud resource utilization and less SLA violations.

This thesis gives an in-depth study on the available statistical forecasting methods, that are applicable on the time series resulting from the monitoring of various application resource requirements over time. Based on literature research we describe how these methods work. The different metrics that are available to determine forecast accuracy are described as well.

The statistical analysis tools are used to provide a model for dynamic forecast selection: to describe what time series features are relevant for the selection process, and how to obtain them. We suggest dividing input time series into three classes based on observed seasonality, trend and variance. For all three classes there are several methods to choose from. To find the optimal method, we split the input trace into test and train subset, apply each of these methods on the train set with respect to past values, and choose the forecast method that has the less forecast errors compared the forecast values with the test set.

This model selection obviously takes a lot of computation therefore we introduce dimensionality reduction techniques to ease the selection process, and even the forecast calculation. This performance improvement is crucial in a large scale cloud system where the system has to classify and forecast on multiple time series from thousands of applications in real time.

A cloud simulator program is implemented into java to evaluate how accurately forecasting algorithms can predict future SLA violations. The results are compared with naive forecasting method like MEAN-forecast, to make sure that the forecasting methods introduced, are outperforming naive approaches. Moreover it is compared to see if the best performing method in the simulation is really the one that is suggested by the classification model.

The simulation runs on synthetic application usage traces, that are generated by workload generator program. The generated trace simulates seasonal cyclic behavior just like the observed real web application traces do.

1.5 Structure Of The Thesis

The thesis is structured as follows: Chapter 1 gives introduction about the topic and why it is important. Chapter 2 outlines the current state of art and the current research efforts in the area. Chapter 3 describes the general architecture of a system that is capable of proactive resource allocation based on monitoring historical application states in the cloud. Chapter 4 compares the available tools to be used for generating workload patterns that are similar to the observed examples. Chapter 5 discuss the possible forecasting methods to be used, and how they work. Chapter 6 debate about selection process to decide which of the methods should be applied for which input data. Chapter 7 shows the implemented simulator to be used for testing the seasonal forecasting methods to early detect sla violations. Finally chapter 8 summarizes the findings and present ideas for future work.

CHAPTER 2

Related Work

Cloud computing is the next generation of computing platforms leveraging from virtualization techniques that allows a pay-as-you go model for renting computational resources to a third party. A cloud typically consists a large number of physical hardware resources and computers that are interconnected through a communication network. With the help of hardware virtualization it is possible to run many applications on the same physical hardware, or alternatively run the same application on many connected computers at the same time. The cloud computing paradigm is gaining increasing popularity in the recent years by offering the advantage that companies and individuals can rent resources from the cloud for storage and other computational purposes on a flexible way. Cloud consumers are able to provision reliable computing resources from the shared pool in a convenient way, without the need for investing and installing the physical hardware by themselves. Infrastructure costs can be reduced significantly.

The services provided by service providers can vary from the infrastructure, platform, or software resources. Each such service is respectively called Infrastructure as a Service (IaaS), Platform as a Service (PaaS), or Software as a Service (SaaS). The scale of cloud infrastructures can vary from small, private, company-owned cloud, to geographically distributed, large publicly accessible, mega-scale infrastructure. Some of the cloud providers are: Amazon, Google, IBM, HP Oracle Cloud, Salesforce, Zoho and Microsoft Azure.

Resource management in cloud systems is an important problem and has been studied extensively. In this thesis we are addressing the problem for online application profiling by using resource usage traces. Online application monitoring and profiling is discussed in the following papers: [22], [41]. There is two general type of trace based resource allocation : reactive and predictive.

2.0.1 Reactive Resource Scaling

Reactive resource scaling is the straightforward way to offer elastic computing services. The idea of reactive scaling is to define scaling conditions based on an arbitrary target metric and reaching the threshold results the addition of extra resources to the cloud automatically. On the opposite way if the inspected target metric is significantly under the threshold, the scheduler may decide to turn down some of the cloud resources.

Many big cloud providers are using rule based reactive scaling process, for example amazon [3]. There are also many studies in this direction how to optimize the reactional scaling process:

AutoScale [30] is one example of a reactive scheduling system, with the focus on keeping spare computing resources turned on to handle sudden bursts. To determine the spare capacity it uses some basic prediction methods: Linear Regression, Moving Window Averaging. Linear regression fits a line equation onto the observed data points so that the overall distance of each point from the line is minimized. Than it is assumed that the future values are going to be near to this line. Linear regression assumes that the trend is going to increase/decrease with the same slope as it did before in the historical traces. Moving window average averages the observations of the last k data points where k is the used window size. It uses this averaged value for the prediction. As the time series is updated with the next observed value(s), the window is updated and the average of the last k values is recalculated. The base of scaling decisions is mostly given by the arrived request rates and QoS metrics such as response time.

[45] describes another reactional scheduling system, with the focus on elastic storage systems. In case when cloud nodes are holding data, the resource scaling becomes a more complex problem. The node can not be turned off even in case of low utilization because if the data is acquired, rebooting the node takes time and leads to terrible delay in response time. Before a low utilization data node can be turned off the data has to be migrated or replicated. Alternatively if a data node is accessed by a large number of users the response may delay due to limitations of other computer resources e.g. CPU. The paper introduces an elastic controller that monitors virtual machine metrics, analyzes the cost of data migration or replication and the impact on the QoS. Than decides when to grow or shrink the data tier and what data to migrate or replicate.

The common challenge of such systems is to deal with highly varying incoming request rates which may cause the scheduler to initiate the turning on and off of hardware resources many times. This is generally undesirable since the booting of servers costs computation resources, takes time, and during the boot time the electricity usage of the computer parts are near to maximum. This also results high risk of amortization [30].

2.0.2 Predictive Resource Scaling

Predictive resource scaling systems are those that are the closest to the topic of this thesis. Such systems are using historical traces to predict future usage and act much earlier than a reactional system would.

PRESS [34] uses short horizon forecasting using markov chains. Markov chains can be used to model traces with high variance. We are going to discuss this method later in Chapter 5. AGILE [54] uses multi-level wavelet decomposition to predict resource usage for low-middle horizon (see Chapter 5). The work described in [5] is focusing on predicting resource usage for web based systems. It is using trend estimation methods to predict future load. It feeds the simple or exponential moving average of the observed traces into ARIMA forecast method.

Aforementioned works so far provide universal methods for trend estimations and short term forecasting for applications. Many existing work found that many cloud applications especially web applications are showing seasonal cyclic resource usage over time. This means that the number of user requests and so the amount of used resources goes up during the day and goes back to minimal level during the night. Predictions based on these seasonal cycles do have good results.

Works described in [33] and [29] using Fast fourier transformation to find long term cyclical workload patterns.

CloudScale [66] is a predictive system that uses FFT based forecasting if seasonal patterns are found in the traces and Markov Chains otherwise. It is also emphasizing the need of adaptive padding: burst based padding adds padding values based on the signal burst pattern, remedial padding makes sure that the system learns from recent prediction errors.

2.1 Application Profiling

Runtime profiling of cloud applications is an effective method to achieve more accurate resource allocation. It is generally true that the more data we collect from the hosted applications the better provisioning decisions can be made. Instead of considering cloud applications as „black boxes“ profiling deals with the problem of collecting in-depth knowledge about how the applications operate. Unfortunately, it is difficult to obtain accurate profile data on live client workloads due to the large overhead of the instrumentation itself. The challenge of the application profiling is to manage the tradeoff between profiling accuracy, perfomance overhead, and the costs incurred for cloud computing platform usage.

In general there are two types of application profiling: online and offline. Online application profiling has the advantage of providing real time data which can be used more effectively for scheduling optimization, but on the other hand it imposes a large overhead to the execution of the application. Example for online profiling is introduced in the work described in [48] . Offline application profiling avoids the problem of the large overhead, but it may lead to inaccurate data collection that does not represent the full spectrum of application execution states [43].

Elastic and adaptive resource usage profiling is introduced in [22]. It allows the real time analysis of system behavior with minimal performance degradation. This is achieved by selectively and adaptively instrumenting only a specific subset of application virtual machine instances. This way realistic profiling data can be acquired about running applications while respecting QoS requirements.

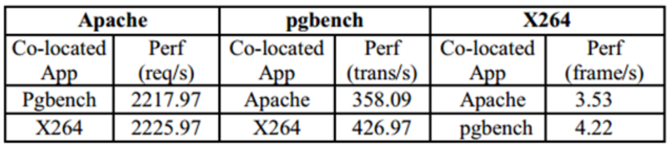
2.2 Monitoring

It is important to decide which application specific metrics one needs to keep track of. In order to allocate the right amount of resources, application specific metrics has to be monitored. The monitoring process has to be configurable: (i) to offer the decision which metrics to monitor, (ii) to scale: to be applicable on any cloud size, (iii) and adapt: by allowing flexible setting of the sampling rates. M4Cloud [50] offers such monitoring capabilities. The author provides a classification about the different application specific metrics. Eight classes are distinguished:

- Application **generic / specific**

- Directly **measurable / calculable** (can be calculated by specific formula using other metrics)
- **Shared** (single measuring mechanisms can be applied for all applications) / **Individual** (measuring mechanisms has to be implemented for each application e.g., number of db entries)
- **Quantity** (amount of resources e.g., CPU usage) / **Quality** (metric that represent a QoS that is guaranteed to be within a threshold e.g., response time)

The collection of different monitoring metrics can happen on different levels: VM-level, PM-level, or Application level. PM-level monitoring may lead to less accurate results as pointed out in paper [22] by the following figure 2.1. Co-locating VM-s with VM-s with different



Apache		pgbench		X264	
Co-located App	Perf (req/s)	Co-located App	Perf (trans/s)	Co-located App	Perf (frame/s)
Pgbench	2217.97	Apache	358.09	Apache	3.53
X264	2225.97	X264	426.97	pgbench	4.22

Figure 2.1: The effect of co-locating different applications on the performance

kinds of workload has different effect on performance. To show the impact, they run three different applications in separate VMs. Apache is a benchmark program for Apache Web server whose performance is measured by the number of requests per second. Pgbench is a benchmark program testing PostgreSQL whose performance is number of transactions per second. X264 is H. 264 video encoding program and its performance metric is number of frames per second.

Application level monitoring leads to very accurate monitoring results, but the monitoring system needs to be adjusted to each application platform separately which is not feasible in a cloud with potentially thousands of hosted applications. VM-level monitoring is applicable in any cloud infrastructure. It does not need customization for each application specifically, and the results of monitoring on applications resource usage is more accurate than in the case of PM-level monitoring.

2.2.1 Forecasting and Prediction for Time Series

Prediction is a major area in several fields of research. Concerning time series, it is one of the most extensively applied tasks. The literature distinguish between prediction and forecasting: prediction is usually used for short horizon forecasting (one step ahead) while forecasting is the term when higher horizon is applied. Generally there are two classes of prediction tools to use: statistical forecasting [39] and machine learning approaches [2].

Statistical forecasting tools are used in this thesis and generally better for revealing inter value dependencies thus can be used for seasonal and trend forecasting. While machine learning approaches are better for tasks where in addition to the dependencies between (lagged) values (inter value dependency/correlation), the forecast depends on other additional variable factors. A good example for the applicability of machine learning prediction is in the case of electricity forecasting: the electricity usage basically exhibits strong regularity (seasonality) and in addition it is dependent from some external factors for example outside temperature, or the day of the week.

Statistical tools, and some machine learning alternatives are introduced more in detail in the chapter 5.

A large-scale comparison for the major machine learning models applied to time-series forecasting is listed in [2]:

- Hidden Markov Model
- Multilayer perceptron
- Bayesian neural networks
- Radial basis functions
- Neural networks (also called kernel regression)
- K-nearest neighbor regression
- CART regression trees
- Support vector regression
- Gaussian processes

Latest research in the area is concentrated on multi-time series forecasting [38] and detecting multiple and irregular seasonality cycles [21].

2.3 Classification

The idea of workload classification and usage of different methods for forecasting based in different signal features comes from [36]. It uses an automatic decision tree to decide which statistical forecasting method to use to forecast the number of users. The methods used by the decision tree are: ARIMA, ETS, TBATS, MA, Cubic Smoothing Splines and Croston's method.

Figure 2.3 shows the result of the workload classification process applied in paper [36]. In every feedback cycle the accuracy of each forecast method is calculated and the best one is chosen. It is written on the bottom which forecast method provides the most accurate prediction for that interval. The example scenario is tested on real data sets that contain the number of users that accessed the wikipedia site of „germany" per hour (24 data points per day). The selection depends on the input characteristics size, variance, periodicity (Workload intensity behavior), the forecasting overhead introduced by the forecasting method, and the accuracy of the forecasting in the last forecast period. The classifier is using direct feedback mechanisms that evaluate and compare the recent accuracy of different forecasting methods.

Next we are describing how to obtain and quantify certain properties of the time series, e.g. seasonality, trend, noise in order to be able to classify them.

2.3.1 Analysis of Seasonality

Seasonal dependency (seasonality) is general component of time series patterns. It is formally defined as correlational dependency of order k between each i'th element of the series and the (i-k)'th element and is measured by autocorrelation (a correlation between the two terms); k

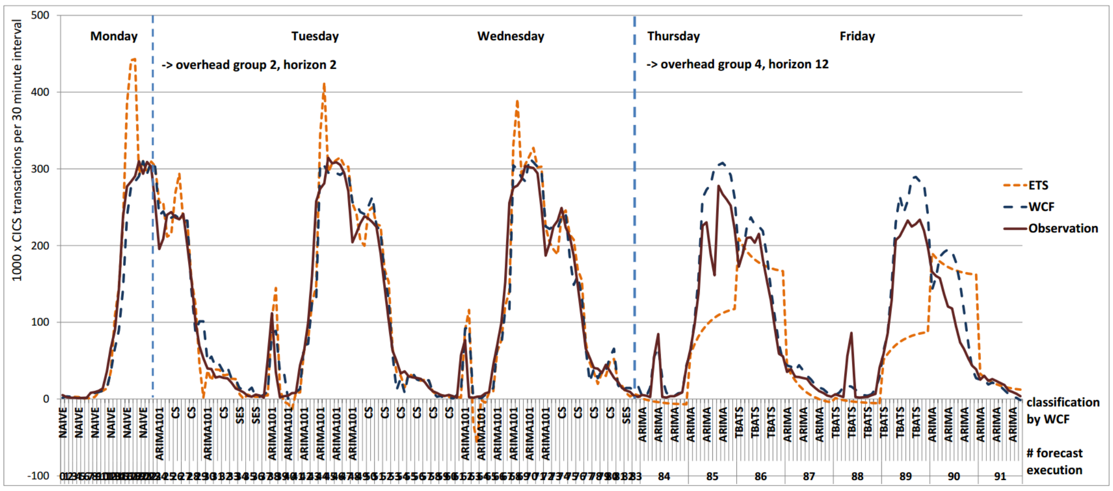


Figure 2.2: The result of the WCF (Workload Classification and Forecasting) approach in [36] compared to exponential smoothing (ETS)

is usually called the lag. If the measurement error is not too large, seasonality can be visually identified in the series as a pattern that repeats every k elements.

Autocorrelation correlogram. Seasonal patterns of time series can be examined via correlograms. The correlogram (autocorrelogram) displays graphically and numerically the autocorrelation function (ACF), that is, serial correlation coefficients (and their standard errors) for consecutive lags in a specified range of lags. Ranges of two standard errors for each lag are usually marked in correlograms but typically the size of auto correlation is of more interest than its reliability because we are usually interested only in very strong (highly significant) autocorrelations.

Partial autocorrelations. Another useful method to examine serial dependencies is to examine the partial autocorrelation function (PACF) - an extension of autocorrelation, where the dependence on the intermediate elements (those within the lag) is removed. In other words the partial autocorrelation is similar to autocorrelation, except that when calculating it, the (auto) correlations with all the elements within the lag are partialled out. If a lag of 1 is specified, there are no intermediate elements within the lag, then the partial autocorrelation is equivalent to auto correlation.

Figure 2.3 shows acf and pacf applied on the Quarterly retail trade data in Euro area. Both suggesting a significant spike at lag 4 [68] [61].

Power spectrum. The power spectrum is the discrete Fourier transformation of the autocovariance function of an appropriately smoothed version of the original series. If one thinks of the time series as sampling a physical waveform, it can be estimated how much of the wave's

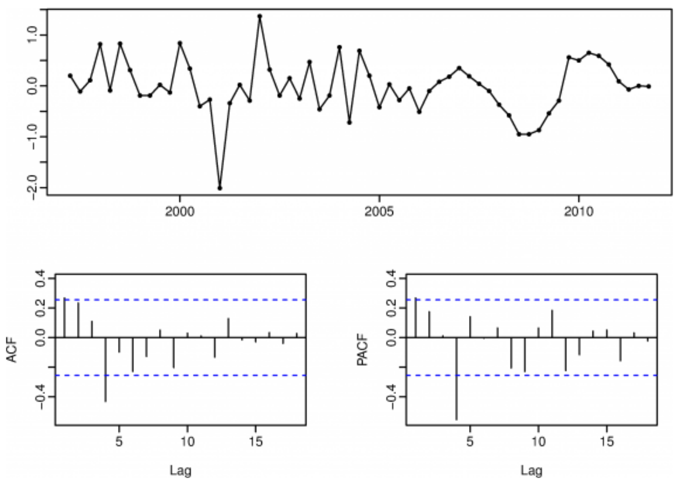


Figure 2.3: ACF, PACF applied on seasonal data

total power is carried within each frequency. The power spectrum (or periodogram) plots the power versus frequency. Cyclic (that is, repetitive or seasonal) patterns will show up as large spikes located at their frequencies [29].

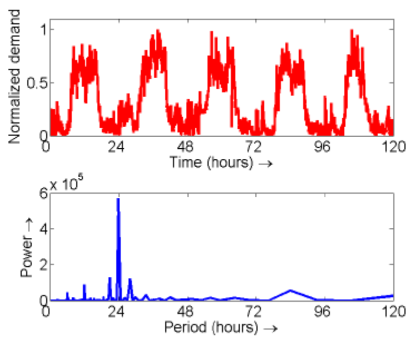


Figure 2.4: Periodogram of seasonal workload pattern

2.3.2 Analysis of Trend

Smoothing. Smoothing always involves some form of local averaging of data such that the nonsystematic components of individual observations cancel each other. The most common technique is moving average smoothing which replaces each element of the series by either the simple or weighted average of n surrounding elements, where n is the width of the smoothing „window“. Medians can be used instead of means. The main advantage of median as compared to moving average smoothing is that its results are less biased by outliers (within the smoothing window). Thus, if there are outliers in the data (e.g., due to measurement errors), median smoothing typically produces smoother or at least more „reliable“ curves than moving average based on the same window width. The main disadvantage of median smoothing is that in the absence of clear outliers it may produce more „jagged“ curves than moving average and it does not allow weighting [68] [13].

2.3.3 Analysis of Noise

Decomposition We decompose the traces into smoothed trends, spiky bursts, and residual data. We apply exponential moving average filter on the raw data. To detect the spikes we determine a threshold for the maximal deviation allowed between the original and the smoothed signal. We call the deviation between the original signal and the smoothed signal the „noise“. An example threshold of the spikes would be three times the standard deviation of the noise signal as shown on the figures below [65]:

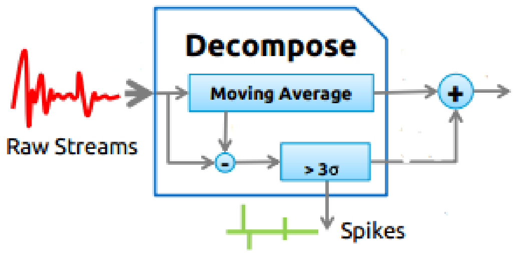


Figure 2.5: Detection of spikes in the signal

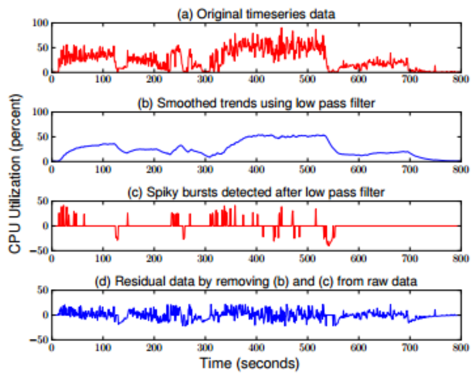


Figure 2.6: Example timeseries decomposition to observe burstiness of CPU monitoring data

This way we can measure the signal burstiness, and divide traces into different classes based on their grade of burstiness.

2.3.4 Workload generation

Workload generators are widely used for testing the performance of Web-based systems. Practitioners and researchers rely on benchmarking systems such as RUBiS, TPC-W and SPECweb to evaluate the performance of their IT infrastructure. The workload generator is a tool that generates a synthetic workload to the benchmark application to emulate the behavior of the application's end users. The workload generator reports metrics such as response times for the emulated users and application throughput. A desirable property of a Web workload generator is that it sends and receives requests to a specified Web server in a realistic manner. To achieve realistic workload generation the generator has to be capable to generate request as it would come from a real user, including the simulation of sessions, inter-request dependencies (the order how the resources of a web application are accessed by the user are usually not random), think time (users need time to process information). The need of realistic workload generator is often ignored in practice, therefore finding a reliable tool that generates realistic workload is not an easy task [59]. We use workload generation for emulating the seasonal behavior of request rates arriving on web application. We compared four existing tools: RUBiS, RAIN, TPC-W, JMeter for this purpose in Chapter 4.

2.4 Cloud Simulation

Quantifying the performance and allocation policies in real cloud computing environment for different application models is extremely challenging. Testing every possible resource allocation strategy is not feasible inside a real cloud, therefore there are many simulators implemented that let researchers compare different strategies and run test without occupying real cloud infrastructure. Cloud Simulators may combine the usage and simulation of hardware and software entities.

2.4.1 CloudSim

One of the most advanced cloud simulator is CloudSim [18] which provides the following set of functionalities:

- simulation of large scale Cloud computing data centers
- simulation of virtualized server hosts, with customizable policies for provisioning host resources to virtual machines
- simulation of energy-aware computational resources
- simulation of data center network topologies and message-passing applications
- simulation of federated clouds
- dynamic insertion of simulation elements, stop and resume of simulation
- user-defined policies for allocation of hosts to virtual machines and policies for allocation of host resources to virtual machines

2.4.2 TeachCloud

Teachcloud is built on the top of CloudSim extending it with additional features such as built in workload generator, Graphical user interface, additional network models, monitoring outlet for most of the cloud system components, dynamic reconfiguration of the cloud to study the impact of changes on the cloud [23].

2.4.3 CDOSim

CDOSim is a cloud deployment option (CDO) Simulator which can simulate the response times, SLA violations and costs of a CDO. A CDO is a decisions concerning simulator which takes decision about the selection of a cloud provider, specific runtime adaptation strategies, components deployment of virtual machine and its instances configuration. CDOSim has ability to represent the user's rather than the provider's perspective. Major advantages of CDOSim are [23]:

- Consequently oriented towards the cloud user perspective instead of exposing fine-grained internals of a cloud platform.
- Mitigates the cloud user's lack of knowledge and control concerning a cloud platform structure.
- Simulation is independent of concrete programming languages.
- Workload profiles from production monitoring data can be used to replay actual user behavior for simulating CDOs.

2.4.4 GreenCloud

GreenCloud aproaches the cloud simulation problem from the aspect of electricity usage. GreenCloud extracts, aggregates and makes fine grained information about the energy consumed by computing and communication elements of the data center equipment such as computing servers, network switches and communication links. The aim is to minimize consumption of electric power by improving power management, dynamically managing and configuring power-aware ability of system devices [23]

Other simulators may examine different aspects of cloud systems. iCanCloud is developed for the simulation of large storage network. It can predict the trade-off between costs and performance of a particular application in a specific hardware in order to inform the users about the costs involved. It focuses on policies which charge users in a pay-as-you-go manner [23].

SPECI (Simulation Program for Elastic Cloud Infrastructures) is another tool which allows analyzing and exploration of scaling properties of large data center behavior under the size and design policy of the middleware as inputs [23].

2.5 Energy Efficient Scheduling

The previously described dynamic cloud scaling systems are mostly concentrated on detecting and reacting on SLA violations. Although this indirectly results better cloud utilization and optimization of power consumption, there are several works that are focused more on the aspect of power consumption optimization. In this section we are giving a brief introduction of scheduling decision methods that are directly taking electricity usage models into account.

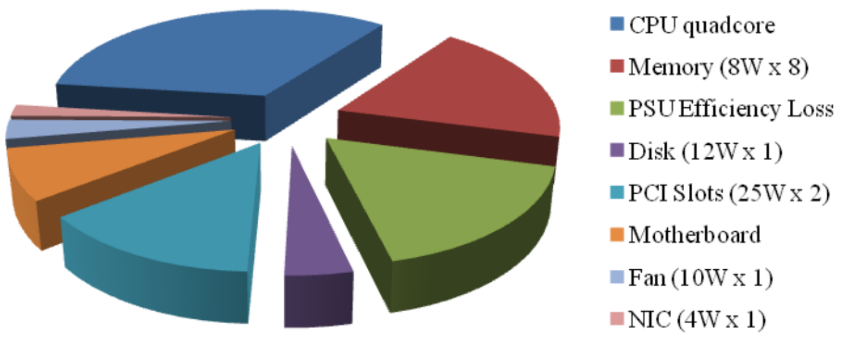


Figure 2.7: Power consumption by server's components

2.5.1 Sources of Power Consumption

The main part of power consumed by a server is drawn by the CPU, followed by the memory and losses due to the power supply inefficiency (Figure 2.7).

Each computer part consumes less in idle state than in peak performance. Modern desktop and server CPUs can consume less than 30% of their peak power at low activity modes, leading to dynamic power range of more than 70% of the peak power. In contrast, dynamic power ranges of all other server's components are much narrower: less than 50% for DRAM, 25% for disk drives, 15% for network switches, and negligible for other components. For a server overall, the difference between the power usage in peak mode, and in idle mode is about 30%. This means that even if a server is completely idle, it will still consume more than 70% of its peak power [10].

According to the article [25] the consumed electricity depends mostly on CPU utilization: the linear model they used is written as:

$$Total system power \approx P_{idle} + (P_{busy} - p_{idle}) \times u \tag{2.1}$$

The energy-efficient scheduling approach in [32] includes different factors in an optimization method and use integer programming in every scheduling cycle to optimize the scheduling based on various strategies. The factors included in the model are: CPU power, CPU frequency, ratio of CPU power consumption to overall power consumption energy cost, CO_2 emission rate, execution price. Generally there are two objectives: minimizing CO_2 emission, and maximizing profit. To optimize this two objectives the paper offers many strategies.

The authors in [44] use genetic algorithm to reduce the cumulative power energy utilized by the system resources while minimizing the execution time for the tasks. The authors formalize the problem as a multi-objective optimization to optimize execution time and energy consumption.

CHAPTER 3

Profiling System Design

In this chapter the architecture of the cloud application profiler is introduced. This profiling system has to be self-adaptive, scalable and applicable in any cloud infrastructure. The task of the profiling system is to monitor the selected system or application metrics, apply forecasting, and make improved scheduling decisions. An important requirement toward the system is that it has to remain flexibly configurable and adapt to continuous changes. In this chapter the profiling process is described, along with the required system components and their jobs.

3.1 Profiling System Architecture

The optimization process consist of five steps: Monitoring the applications, classifying the application usage traces, calculating the forecast values, analyzing forecasts and finally making an optimized assignment of applications to the available resources [41].

The optimization process consist of five steps: Monitoring the applications, classifying the application usage traces, calculating the forecast values, analyzing forecasts and finally making an optimized assignment of applications to the available resources. The system is supposed to implement a feedback loop, since, based on the analysis of the forecast values and patterns, it not only has to make scheduling decisions but also has to control the further monitoring process. This loop will ensure that the system continuously adapts to the changing states of the cloud. The profiling should be able to facilitate monitoring and collecting various kinds of measures on different levels: Hypervisor-level, VM-level, Application-level. This thesis explores basic resource usage metrics of an application on the VM-level such as: The optimization process consist of five steps: Monitoring the applications, classifying the application usage traces, calculating the forecast values, analyzing forecasts and finally making an optimized assignment of applications to the available resources. The system is supposed to implement a feedback loop, since, based on the analysis of the forecast values and patterns, it not only has to make scheduling decisions but also has to control the further monitoring process. This loop will ensure that the system continuously adapts to the changing states of the cloud. The profiling should be able to

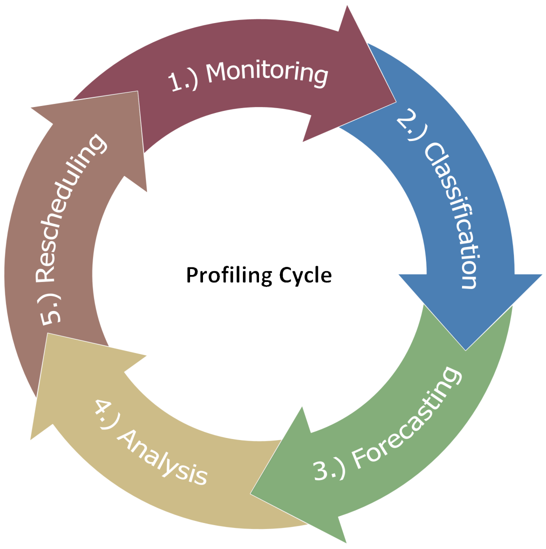


Figure 3.1: Process Stages

facilitate monitoring and collect various kinds of measures on different levels: Hypervisor-level, VM-level, Application-level. This thesis explores basic resource usage metrics of an application on the VM-level such as:

- CPU performance (%)
- Memory (MB)
- I/O operations (Read bytes, Written bytes per seconds)
- Network traffic (KB/s)

In the further steps an analysis on the collected data is conducted. The analysis attempts to find patterns in each resource usage that will help to predict and forecast future resource needs and adapt the system for the next profiling loop.

In today's scale, a cloud cluster can contain many hundreds of computational nodes, each equipped with multi-core processors and enough additional resources to host many virtual machines. In this environment it is important to provide a scalable architecture that can monitor and calculate forecasts, and control the cloud accurately, without imposing large additional costs. A sketch of the distributed profiling architecture is shown in Figure 3.2.

The cluster consists of three types of nodes:

- **Master Node**
  The Master Node controls the monitoring processes on the Slave Nodes, they aggregate the data and facilitate distributed calculations.

- **Slave Node**
  Slave Nodes host the virtual machines on which various kinds of applications are running. Additionally each VM will be equipped by a monitoring application.

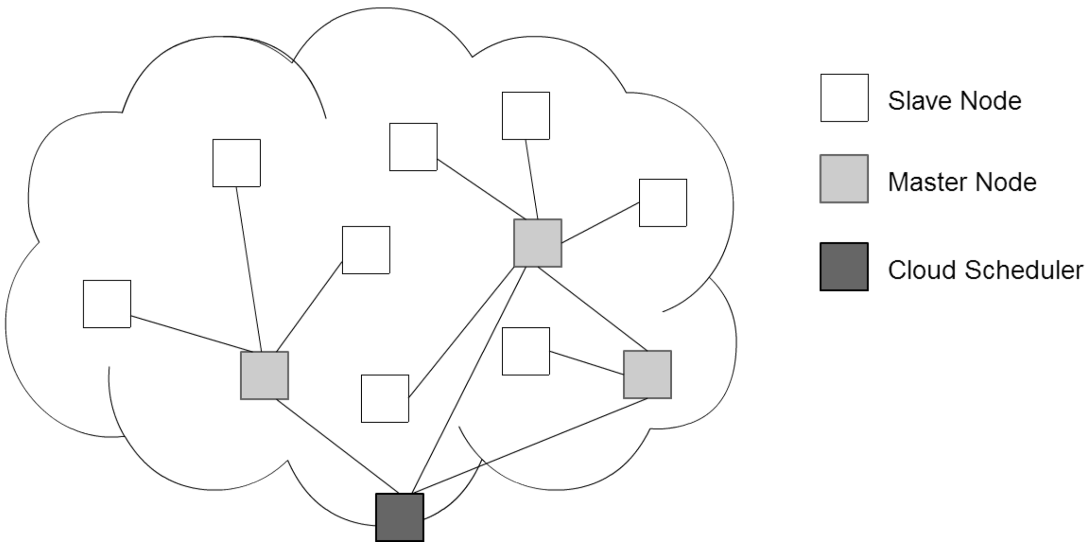


Figure 3.2: Distributed Profiling

- **Cloud Scheduler**
  The Cloud Scheduler has access to all Master Nodes. Based on the forecasts calculated by the Master Nodes it makes scheduling decisions and controls the profiling process.

3.1.1 The System Components

Figure 3.3 represent the component diagram of the system. Each component will be described in more detail in later chapters. That will help to explain how the system works:

- **VM Level Resource Monitor**
  To each VM an additional agent is going to be deployed. This agent is controlled by exactly one other master node. The master node determines which metrics the agent has to collect and what time intervals it should use. First, the data is stored in a local database, and it is only sent to the responsible master node.

- **Slave Controller**
  The master nodes have all the knowledge about the monitoring policies applied on the different VMs. To keep the monitoring process coordinated and also to prevent unnecessary monitoring overhead, the master nodes are responsible to send the right commands to the monitoring agents: how long they are supposed to collect data, what time interval they should use, which resources to log, etc.

- **Data Acquisitor**
  Data acquisitor collects all the data from the slave nodes, and stores it in a common Time

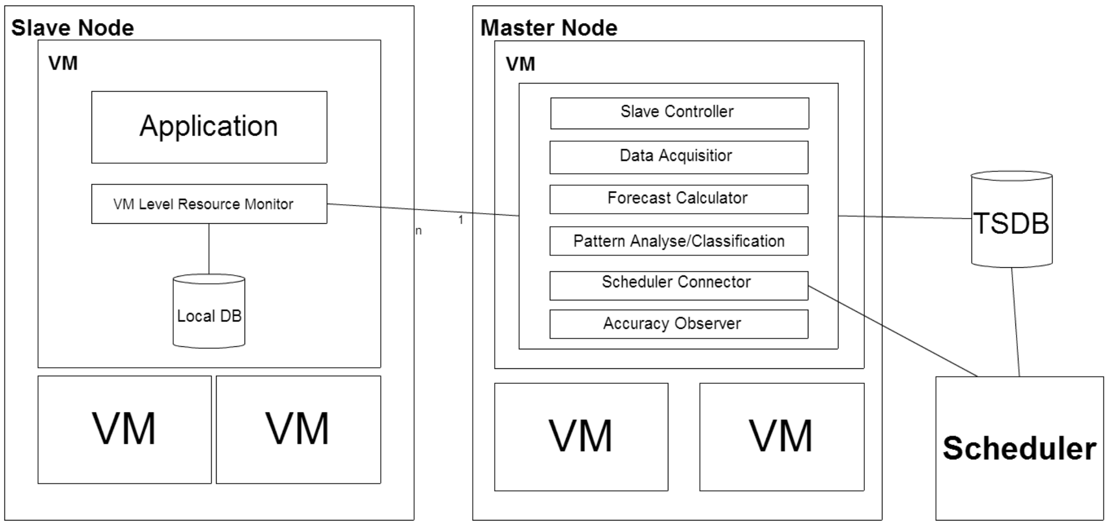


Figure 3.3: Component Diagram of the profiling system

Series Database (TSDB). To store the data in a common TSDB's has an additional advantage: the data is ensured to be in a common format. Many TSDB's offer out-of-the-box visualizations, compressing mechanisms and additional features. Most importantly master nodes can work on the same data in distributed manner: the master node that has free resources in a given time can work on resource intensive calculations concerning forecasting, classifying or analyzing.

- **Pattern analyze/classification**
  The Pattern analysis and classification component includes the tools and techniques to divide the application into three classes based on forecasting objectives:

  - Seasonal class: Applications with seasonal cycles
  - Trend interpolation class: Applications with trend component
  - High variance class: The rest of applications with highly volatile changes in resource needs

  In addition to classification this component has the responsibility of selecting the best performing forecasting method in each class, adjusting forecast periods, and forecasting horizon or confidence levels, if needed.

- **Forecast Calculator**
  The Forecast Calculator component has a collection of algorithms that calculate the forecast values based on sophisticated models. Based on our experiences we found that calculating the forecast values requires more computational cost depending on the size of the

input data set and the forecast horizon size. This thesis focuses on working with statistical time series analysis techniques.

- **Accuracy Observer**
  The accuracy observer observes the difference between the actual and the forecasted resource usage. If the error rate exceeds a certain threshold, dynamic error correction is applied, or reclassification of the observed traces is initiated.

- **Scheduler**
  The scheduler is the ultimate controlling entity in the cloud which makes the optimal resource allocation decisions based on the data and then propagates the decision to the master nodes, which execute the scheduling decisions.

- **Master Node**
  The master node receives the monitoring data from slave nodes and analyzes them. In a large scaled cloud there are potentially many master nodes necessary to handle the profiling process of each cloud applications.

- **Slave Node**
  Slave nodes are hosting the cloud applications and monitoring the VM-level resource usage metrics, for example cpu or memory usage.

3.1.2 Monitoring

There are many levels from which resource usage information can be extracted: Physical Machine-level, VM-level, Application-level. Usually the deeper we go the better accuracy we can achieve [22] [50] [74]. Monitoring the physical machine level does not provide enough information since it is not possible to extract application resource usage patterns caused by the applications internal processes. Aggregated resource usage can only be obtained from the results of the resource needs of many co-hosted applications and virtual machines. Monitoring the application directly, is on the other hand very accurate, but the monitoring then has to be adjusted to the applications platform. This customization is a big implementation overhead. As a tradeoff, the suggested systems could facilitate a VM-Level Resource Monitor.

Accuracy. The Profiling system should be able to change the monitoring accuracy based on the requirements given by the feedback loop or the system administrator. The levels of accuracy shall be defined as follows:

- **Level 1**: The perfect condition to monitor an application is to let it run on a separate physical machine and separate VM that has no other workload, monitor the VM's system resources and subtract the previously measured basic idle system resource usage.

- **Level 2**: Allow the PM co-host more than one virtual machine. This way is less accurate because the virtual resources may interact less efficiently with the real resources.

- **Level 3**: One can monitor each application process separately and aggregate the result. This approach is the least accurate because application processes often call up kernel functions which cause additional resource usage by the kernel and not the actual process.

In addition to accuracy levels, it is also crucial which measurement intervals the system chooses to use. If a small time interval is chosen, then one faces a larger performance overhead. If it is too large important details in the forecasts may be missed. Interval which are too short produce too much data, intervals which are too long interval can cause aliasing.

The measurement intervals should match with the forecasts objectives. If one wishes to forecast the long term, meaning that the usage exhibits daily seasonal patterns, one may want to set the intervals to a significantly large value. If the forecast objective is to forecast middle range trends, a somewhat finer granularity is needed to catch the trend changes. If the usage pattern is highly volatile it is still possible to try to forecast the short term by employing various methods described later. When forecasting in the short term, the shortest measurement intervals are necessary to catch the changes in an application internal phases.

Metrics. The profiling system has to offer flexibility to choose which metrics the system administrator wants to monitor and forecast. Here basic resources were chosen to be monitored: cpu, memory, disk I/O, network download/upload. There are potentially many other metrics to consider: disk free space, swap size, distinguish between internal/external network usage etc. Another issue with the monitored resources is that they cannot be expected be equal among the cloud cluster. For example, the CPU usage is measured in percentage of the core usage, but this percentage has a different meaning in the case of diverse processors. One idea to overcome this difficulty is to use existing processor benchmark data (e.g: www.cpubenchmarks.com) and adjust the percentages that are measured on different CPU models according to the benchmark data proportions.

3.1.3 Classification

Classification is the process which divides the measured application resource usage time series into one of the following classes based on forecastability attributes:

- **Seasonal class**: Resource usage Time Series that have seasonal cycles
- **Strong trend class**: Resource usage Time Series that do not have seasonal cycles but have a steady trend component
- **Short time prediction class**: Resource usage Time Series that are fluctuate highly and do not have seasonal cycles, nor steady trends.

3.1.4 Forecasting, Analysis, and Rescheduling

The reason for the classification is to apply the most appropriate forecasting method to the time series properties. For each class there are several forecasting methods applicable, the table below (3.4) summarizes the suggestions. To improve the results of the forecasting further trace analysis can be applied:

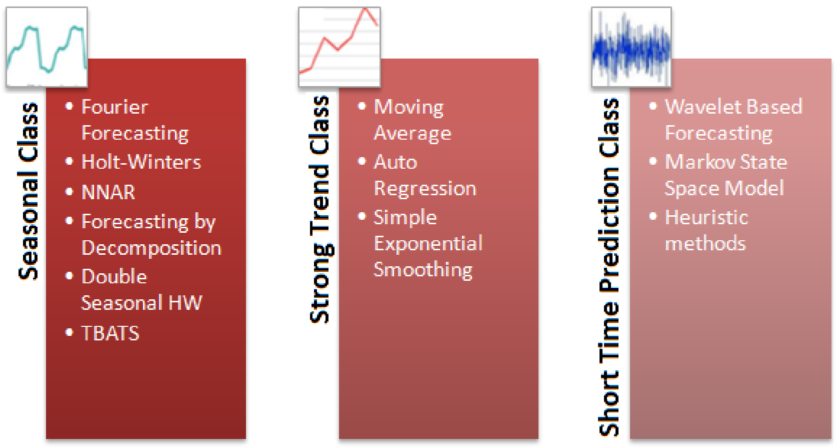


Figure 3.4: Based on the properties of the traces (seasonality, trend, noise) we divide them into three classes. Each class has many applicable forecasting methods that are strong at forecasting time series with certain properties. The selection of the best performing forecast method is made by benchmarking them.

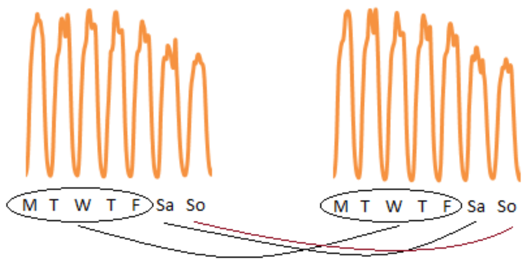


Figure 3.5: Determining days with similar usage. Traces extracted from two consecutive weekly accesses to the „Germany“ wiki page.

Day-of-the-week analysis. Based on the observed usage traces, it can be seen that the usage cycles of some days-of-the-week are more similar to others. For example usually the cycles of the workdays are more similar to each other in magnitude as are the weekend days to each other. During the analysis process it is important to calculate which days are similar, because it helps to improve the accuracy of the forecasts. Previously similar days can be used as the input for the forecast.

Time Series Similarity. To build these similarity graphs a reference day is taken and a similarity measure is calculated between the reference day and the other days. Similarity Measure: D(T,U) between time series T and U is a function taking time series as inputs and returning the distance d between these series. An example of distance metrics is the Euclidean distance

$$d(p,q) = \sqrt{\sum_{i=1}^{n}(q_i - p_i)^2}$$

q_i : The i-th datapoint of the first signal
p_i : The i-th datapoint of the second signal

The time series whose distances are measured to be closest to each other are most similar [24].

Benchmarking Algorithms. An essential part of this process is to offer an automatic evaluation framework that selects the best performing algorithm from the repertoire. In order to achieve this, the classification process calculates the forecast values based on each different method and selects the one that is proven to be most accurate. The accuracy is measured by splitting the historical traces into a train and test set, calculating the forecast based on the train set and comparing the difference between the forecast values and the test set.

Rescheduling. Rescheduling improves the resource allocation strategy in a proactive way. The ultimate goal is to distribute the applications on a minimal set of physical machines without violating the boundaries of the available resources.

CHAPTER 4

Workload Generation

In this chapter the problem of synthetic resource usage trace generation is discussed. Generated traces are used to test the forecasting methods on and the same traces are used in our cloud simulation. In order to generate proper traces that are similar to that, what a common cloud application would result one must first observe some real world examples. Then some of the available tools that can be used for workload generation are described and compared. At the end of the chapter the traces that were generated are plotted and described.

4.1 Observations

In order to develop, compare and test the forecasting methods, one needs test data. This test data will contain application traces that are collected during multiple days. Each application is represented by multiple resource dimensions (cpu, net, ram, disk i/o) that was needed in the given time span. The basic idea is to generate workload on a virtual machine and monitor the resource usages.

While observing different traces of user accesses on different kinds of applications, it is seen that most of the time web based applications tend to draw similar patterns:

Patterns described in Figure 4.1 are collected from the Web server at the University of Waterloo, a department-level Web server at the University of Calgary, a campus-wide Web server at the University of Saskatchewan, the Web server at NASA's Kennedy Space Center, the Web server from ClarkNet, a commercial Internet provider in the Baltimore–Washington, DC, region, and the Web server at the National Center for Supercomputing Applications (NCSA) in Urbana-Champaign, IL. Although the servers are scaled for different number of users the pattern holds steady: the number of request arriving are at the minimum at around 5 am and they are peaking in the early afternoon.

And the same pattern tends to hold with little deviation from day to day:

The workload demands are very bursty in nature and often vary significantly during the day. The key observation is that despite the big variance the pattern is similar from day to day.

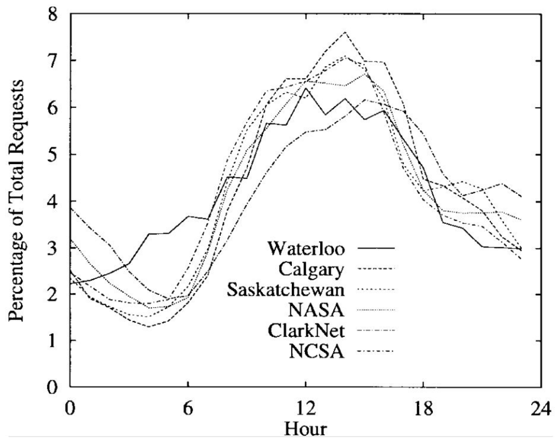


Figure 4.1: Distribution of hourly request arrival rate by server [6].

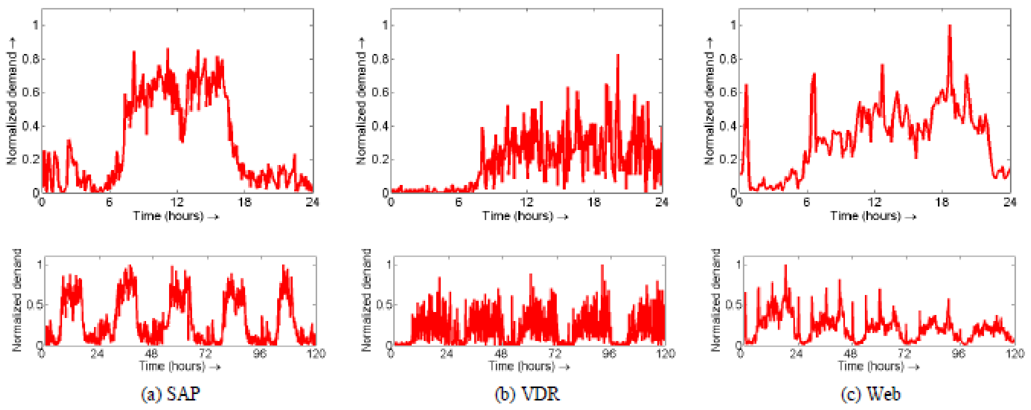


Figure 4.2: A 24-hour and 120-hours demand trace for three real-world traces: an SAP enterprise application, a multi-tier business application called VDR, and a web application. The demands have been normalized by the maximum demand in each trace. [29]

The pattern only breaks at the weekends as shown by the wikipedia dataset in the Appendix: A.5. Other traces that confirm this pullback at the weekends are: worldcup98, internet service provider, and clarknet data (see Appendix A.1, A.2 A.3, A.4).

It is assumed that each request causes some amount of increase in resource usage on the server and that one can expect that the resource usage that is observed, will draw similar curve. Although, the proportion of the used resources is typically different for different types of applications in the cloud. Moreover the daily change of the resource usage is different too. However the seasonal behavior can not be expected for all the applications: majority of web based applications are expected to have seasonal cycles based on the observed data. Also Applications that run batch scripts at fixed time of the day are going to have also seasonal peaks. Yet, seasonal behavior can not be expected from batch/high performance applications that are randomly triggered by users.

This chapter describes the way the test data was collected and provides a comparison of the different workload generation tools that can be used for this purpose.

4.2 TCP-W

TPC-W is a standard benchmark that models online bookstores. The online bookstore workload is a classical client-server webapplication with a database. TPC-W simulates three different profiles by varying the ratio of browse to buy: primarily shopping (WIPS), browsing (WIPSb) and web-based ordering (WIPSo). The primary metrics are the WIPS rate, the associated price per WIPS ($/WIPS), and the availability date of the priced configuration. The benchmark's main feature includes [19]:

- Multiple online browser sessions,
- Dynamic page generation with database access and update,
- Authentication through secure sockets layer (SSL) version 3 or transport layer security (TLS),
- Payment authorization through an emulated payment gateway emulator (PGE), which is not part of the SUT,
- Databases consisting of multiple tables with a wide variety of sizes, attributes, and relationships,
- Enforcement of ACID properties (atomicity, consistency, isolation, and durability) on database transactions, and
- Online transaction execution resulting in contention on data access and update due to concurrency [31].

The TPC-W project is obsolet as of 4/28/05 [19].

4.3 RUBiS

RUBiS: Rice University Bidding System is an auction site prototype modeled after eBay.com that is used to evaluate application design patterns and application servers performance scalability. It is implemented in three different technologies: PHP, Java Servlets and Enterprise Java Beans (EJB) so that developers can compare these alternatives. As a database RUBiS uses MySQL that cotains 7 tables and can be loaded with up to several hundred thousand entries.

It comes with a Benchmarking tool that emulates users behavior for various workload patterns and collects statistics. The auction site defines 26 interactions that can be performed from the client's Web browser. Among the most important ones are browsing items by category or region, bidding, buying or selling items, leaving comments on other users and consulting one's own user page (known as myEbay on eBay). Browsing items also includes consulting the bid history and the seller's information. Two workload mixes are defined: a browsing mix made up of only read-only interactions and a bidding mix that includes 15% read-write interactions. The bidding mix is the most representative of an auction site workload, and it implements a client-browser emulator. A session is a sequence of interactions for the same customer. For each customer session, the client emulator opens a persistent HTTP connection to the Web server and closes it at the end of the session. Each emulated client waits for a certain think time before initiating the next interaction. The next interaction is determined by a state transition matrix that specifies the probability to go from one interaction to another one. The think-time and session time for all benchmarks are generated from a negative exponential distribution with a mean of 7 seconds and 15 minutes, respectively. The load on the site is varyed by varying the number of clients [70].

4.4 Rain

Rain is a statistics-based workload generation toolkit that provides a thin, reusable, configuration and load scheduling wrapper around application-specific workload request-generators, which can easily use parametrized or empirical probability distributions to mimic different classes of load variations. The load scheduling harness takes care of managing variations in the amount of load and the mix of operations, while application-specific request generators are responsible for adapting to changes in the mix of operations and simulating data hotspots (if required).

The big advantages against previous systems is that RAIN can use many previously adapted workload and with little effort developers can adapt any kind of workload to use as a base. The general architecture is illustrated in Figure 4.3.

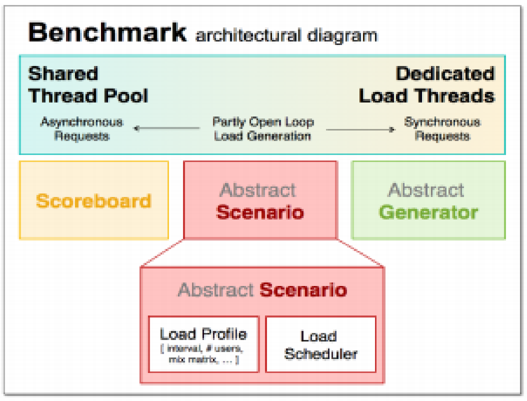


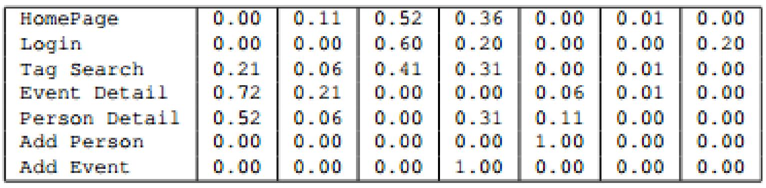
Figure 4.3: Rain Architecture [9]

The flexible architecture allows a wide variety of different configuration: one benchmark run utilize three basic components.

The *Scoreboard* entity is a persistent storage for the execution results.

The *Scenario* entity contains all the configuration parameters for an experiment, including, but not limited to, the maximum number of users to emulate, the duration of an experiment, the ramp up and ramp down interval and the sequence of *LoadProfiles* to enact during a run.

LoadProfile contains an interval (in seconds), the number of users active and the name of a *mix matrix* describing the behavior of each user. A *mix matrix* gives a transition probability between the different operations that the threads are emulating. An example *mix matrix* is illustrated in Figure 4.4 with the probability of the user navigating from a site *A* to a site *B*. During the experiment sequences *LoadProfiles* allow one to vary the number of users and their behavior over time.



HomePage	0.00	0.11	0.52	0.36	0.00	0.01	0.00
Login	0.00	0.00	0.60	0.20	0.00	0.00	0.20
Tag Search	0.21	0.06	0.41	0.31	0.00	0.01	0.00
Event Detail	0.72	0.21	0.00	0.00	0.06	0.01	0.00
Person Detail	0.52	0.06	0.00	0.31	0.11	0.00	0.00
Add Person	0.00	0.00	0.00	0.00	1.00	0.00	0.00
Add Event	0.00	0.00	0.00	1.00	0.00	0.00	0.00

Figure 4.4: Example transaction matrix [9]

The Generator creates requests for a single active user.

Rain is a very sophisticated workload generator with lot of adjustable settings, and it is used as part of the *VMware VMmark Virtualization Benchmark System* for benchmarking cloud systems [9].

4.5 JMeter

JMeter is an application designed to load test functional behavior and to measure performance. It may be used to test performance both on static and dynamic resources like files, web dynamic languages: PHP, Java, ASP.NET, java objects, databases and queries, FTP servers and more. It can be used to simulate a heavy load on a server, group of servers, network or objects to test their strength or to analyze overall performance under different load types.

With a little creativity it can be used as a very flexible workload generator as illustrated in Figure 4.5 [27].

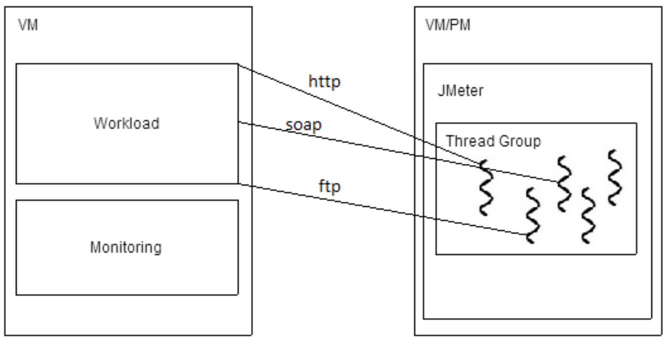


Figure 4.5: Workload generation with JMeter

Figure 4.5 shows a virtual machine with an arbitrary workload. From an other machine we are using JMeter to generate threads that are simulating users behavior and sending requests on the hosted workload. Additionally, we using existing plugins [28] to gain more control on the process:

- **Markov4JMeter** [37]

  Each thread simulates a user who accesses the workload. **Markov4Jmeter** allows the definition of a transition probability matrix: what is the probability that action A is followed by action B. Moreover it allows to add randomized waiting periods between the operations, which makes the whole test more realistic since users are waiting between certain operations too, and the wait time is random within certain boundaries, where these bounds are unique for each operation. The plugin is capable to define any number of transition probability matrices, where each represents a user profile. The probability of that a newly created thread emulates a certain user profile can also be customized.

- **Ultimate Thread Group Plugin**

  Ultimate Thread Group Plugin allows to take full control on the number of created threads along the workload generation.

While generating a workload on the virtual machine it is important to monitor the resource usage. The main advantage of this type workload generation is that one can use any kind of workloads, not just certain web applications. The communication with the workload does not even has to be http-based, one can use soap, jdbc, jms and many other protocols. This way one can directly monitor multiple consolidated applications too.

4.6 Workload Generation Summary

	Workload Complexity	Interaction Complexity [1] [16]	User/Load Profiles	State Transition Matrix	Think-Time Simulation
TPC-W	Simple	Medium	No	Fixed	No
RUBiS	Simple	Simple	Yes	Configurable	Yes (Fixed Random Function)
RAIN	Variable	Variable	Yes	Configurable	No
JMeter	Any	Any	Yes	Configurable (with plugin)	Yes (Configurable Random Function)

Table 4.1: Comparison of the investigated workload generators

[1]Comparison of the data types and amount that are exchanged during a user call e.g. html, js, css, multimedia files.

The main differences of the workload generators are summarized in Table 4.1. For the workload generation we used the JMeter approach. The crucial point is that it can be used with any kind of workloads and additionally consolidated applications can be directly investigated. JMeter approach has an additional advantage with applications that are spread out on multiple VM Nodes. In theory this approach allows to avoid the need to monitor each of these VM nodes: In order to get the usage traces of the application, instead directly monitoring each VMs, we monitor the number of users (this takes less resources than to monitor each VM) and track the users behavior (to construct transaction matrix, and determine the waiting times). Once we have these information we recreate the same requests in a smaller scale: we reduce proportionally the number of users, reply the requests, monitor the resources, than multiplying these traces back gives us the needed data. However the ultimate thread group plugin does not allow the dynamic creation of threads defined by a statistical function in order to simulate variable load intensity with certain statistical properties: e.g, complex-seasonality.

4.7 Generated workload

In the first tests JMeter with Drupal workloads are used. Drupal is a widely used content management system written in PHP. We monitor the *CPU, RAM, Network, Disk I/O* usage on the VM and additionally we inspect the changes in the response time. In the first test we increase the number of threads/simulated users stepwise and we monitor the characteristics of the resource usages and the response times. Figure 4.6 shows the results.

From the outcome we can conclude that:

- CPU and Network usage are strongly correlating, the addition of extra threads cause clear increase in usage level. These metrics have high variance.
- Memory increasing stepwise as JMeter starts additional threads. The variance is minimal.
- Response times are increasing proportionally with the load intensity. Although after reaching the 100 % cpu usage limit (marked by circle in Figure 4.6) the response times are becoming highly changing. Before reaching the limit, response times are below 800ms but after exceeding the limit the response times are sometimes much higher.
- Disk I/O usage is sporadic: Disk I/O operations are not taking place in each monitoring interval. During some intervals it is zero and during other intervals a spike in the data can be seen.

The trace we are generating represent one day. In this day the number of users accessing the website (with different types of requests) goes gradually up during the day. It reaches a peak and than decreases gradually. Similar trend can be seen in other observed data (wikipedia, worldcup98 traces, etc, see appendix A). Figure 4.7 shows the number of running threads over the time. Figure 4.8 shows the resulted usage traces.

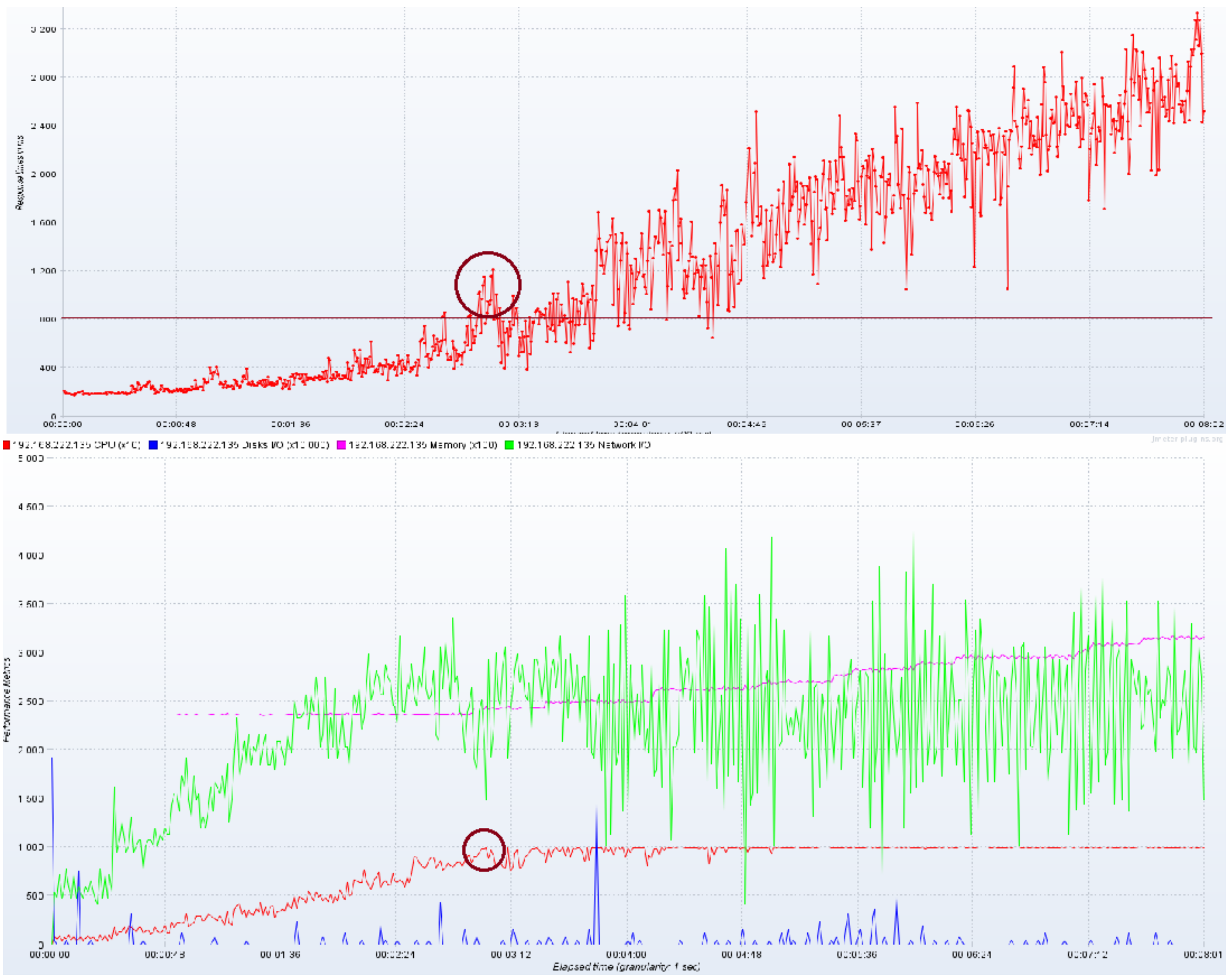


Figure 4.6: On the top: response times in ms, On the bottom: Normalized resource usages: Red - CPU; Green - Network; Pink - Memory; Blue - Disk I/O

- Per-second monitoring is performed on the virtual machine for 24 minutes. This will produce 1440 data point. This data is used as it would be for one day period: there is a data point for ever 5 minutes interval during a one day period.
- One can concatenate one day traces with some amount of distortion. Based on the observation that consecutive days have slightly different amount of traffic (e.g. wiki traces A.5).
- It can be assumed that the collected traces of applications were made on at least two consecutive days. The statistical analysis tools are applied to extrapolate/forecast future resource usage, having at least two cycles is necessary for seasonal forecast methods.

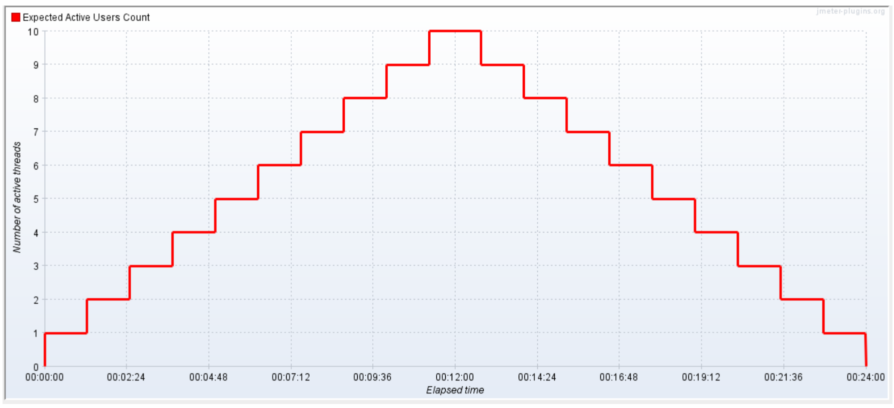


Figure 4.7: Number of threads over time

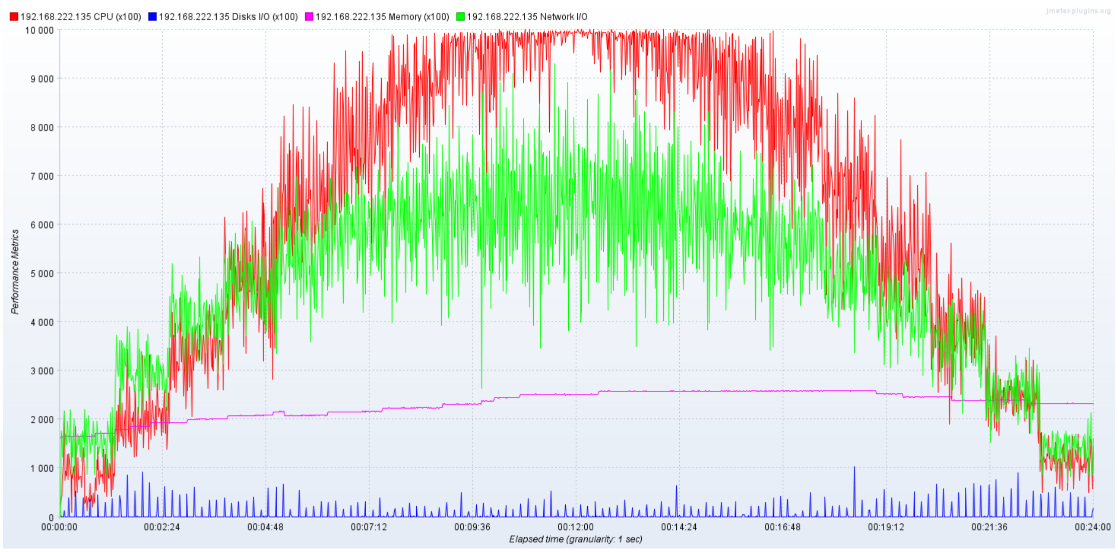


Figure 4.8: One cycle of generated resources (Normalized): Red - CPU; Green - Network; Pink - Memory; Blue - Disk I/O

CHAPTER 5

Forecasting

This chapter presents the available and used forecasting tools. The way they work and under what circumstances they are expected to perform the best are described. After explaining some basic characteristics and definitions in time series analysis, the trend interpolation methods are discussed, seasonal forecasting methods and methods that can be used for short horizon prediction. This chapter also describes how forecast accuracy can be measured, which available metrics there are for this purpose and what their strengths and weaknesses at measuring accuracy are.

5.1 Introduction

The division of monitoring traces into forecast classes during the monitoring feedback loop is explained in chapter 3. In order to apply the best forecasting strategy and tool, the traces will be divided into three classes based on their signal characteristics. They will be classified in seasonal, strong trend and short time prediction class. For each class a different set of forecasting methods and different forecasting strategy is used: seasonal forecasting, trend interpolation, short horizon prediction. The forecast horizon, how many steps ahead the method forecasts is also variable per each class, and each class requires variable prior data length.

Seasonal class. Time Series with strong seasonal component. For this type of forecasting method long horizon forecasting can be expected to deliver accurate results. The long scale means the horizon expands to one or more seasonal cycle(s) ahead. In case of observed daily seasonality the forecast horizon is the next day. Methods include:

- Forecasting By STL Decomposition
- Holt-Winters Exponential Smoothing
- Seasonal ARIMA
- Neural Network Autoregression
- Fourier Forecasting

- TBATS
- Double Seasonal Holt-Winters

This thesis focuses on this forecast class, since the majority of observed traces are proven to be seasonal, although not every application in the cloud is similar and that is why the prior classification is needed. An advantage of this class is that these methods provide a long scale forecast which helps to optimize the scheduling process long in advance. Disadvantages are that the data of at least two seasonal cycles (if the seasonal period is not fixed) has to be previously collected, and due to the larger input and the larger horizon the calculations are more costly (than the methods used for forecasting the other classes).

Strong trend class. The further division between strong trend class and short time prediction class is based on the amount how fluctuating the time series are. Finding the optimal threshold for this division remains part of future work, we refer to these time series intuitively as less-, and more- fluctuating time series. Strong trend class is the class for less fluctuating time series. Trend estimation methods will be used to extrapolate time series on medium scale. Methods include:

- Linear Regression
- Autoregression
- Moving Average
- ARIMA
- (Simple) Exponential Smoothing

These calculations are less costly (than the ones for seasonal class) but they have to be executed more often. Advantages are that the horizon and input data can be flexible, and these methods can be used just somewhat after the monitoring process is started (do not need whole seasonal cycles to be monitored previously). A disadvantage is that there are certain errors on trend changes.

Short term prediction class. For fluctuating time series the proposed system could leverage from methods described in [34] and [54].

- Markov Chain Model
- Wavelet based demand prediction

These methods use short input and deliver short scale predictions.

5.2 Prerequisites

In this section we introduce the basic definitions and concepts needed for time series analysis.

Time Series. A time series is a set of observations made sequentially in time, denoted set by either $x_t : t \in T$ or $x(t) : t \in T$. Time series can be continuous or discrete. In this thesis - since the monitoring process collects measurements made at uniformly spaced time instants given a predefined sampling rate - we are dealing with discrete time series. Thus if the index of the measurement is known and so is the sampling rate, the exact time of the measurement can be calculated. (e.g. sampling rate is h then x(n) represent data collected at the time $T = n * h$).

Time Series Decomposition. There are three basic types of time series patterns:

- **Trend**
  A trend exists when there is a long-term increase or decrease in the data. It does not have to be linear. Sometimes we will refer to a trend "changing direction" when it might go from an increasing trend to a decreasing trend.

- **Seasonal**
  A seasonal pattern exists when a series is influenced by seasonal factors (e.g., the quarter of the year, the month, or day of the week). Seasonality is always of a fixed and known period.

- **Cyclic**
  A cyclic pattern exists when data exhibit rises and falls that are not of fixed period, but (seemingly) random periods. The fact that the cyclic patterns have random periods distinguishes them from seasonal patterns.

STL is an acronym for "Seasonal and Trend decomposition using Loess", while Loess is a method for estimating nonlinear relationships [17].

The time series T shall be thought of as comprising three components: a seasonal component, a trend-cycle component (containing both trend and cycle), and a remainder component (containing anything else in the time series). For example assuming an additive model we write:

$$x_t = S_t + T_t + E_t$$

where x_t is the data at period t, S_t is the seasonal component at period t, T_t is the trend-cycle component at period t and E_t is the remainder (or irregular or error) component at period t. Alternatively, a multiplicative model would be written as:

$$x_t = S_t \times T_t \times E_t.$$

STL Decomposition of the generated test data, that represents the daily CPU usage in percentage for 3 days period using 5 minutes sampling rate is shown the figure below 5.1. A clear seasonal component can be observed and the trend is increasing steadily over the three days period [61].

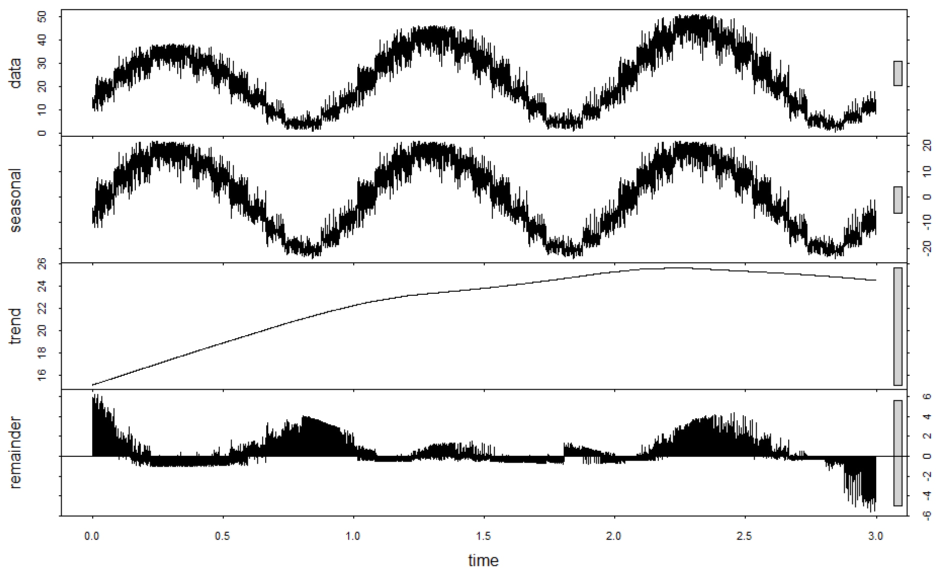


Figure 5.1: STL Decomposition of test data using R

Time Series Forecasting. The time series forecasting problem is about to predict predict the k next values $(t_{n+1}, ..., t_{n+k})$ that are most likely to occur, given an input time series $T = (t_1, ...t_n)$. k is called the horizon of the forecast. There are two basic methods of forecasting: either provide a rule to calculate the next forecast values (like weighted averaging the input) or try to model the possible stochastic process that may generated the time series that we are inspecting and use the model to calculate future points in the time series.

Time Series model. A time series model for the observed data x_t is a specification of the joint distributions (or possibly only the means of covariances) of a sequence of random variables X_t of which x_t is postulated to be a realization. Often the stochastic process (model), that generated the time series to calculate forecast values is used.

5.3 Trend Forecasting

5.3.1 Linear Regression

Linear regression attempts to model the relationship between two variables by fitting a linear equation to observed data. One variable is considered to be an explanatory variable, and the other is considered to be a dependent variable. For example, a modeler might want to relate the weights of individuals to their heights using a linear regression model. A valuable numerical measure of association between two variables is the correlation coefficient, which is a value between -1 and 1 indicating the strength of the association of the observed data for the two variables. A linear regression line has an equation of the form

$$Y = a + bX$$

, where X is the explanatory variable and Y is the dependent variable. The slope of the line is b, and a is the intercept (the value of y when $x = 0$).

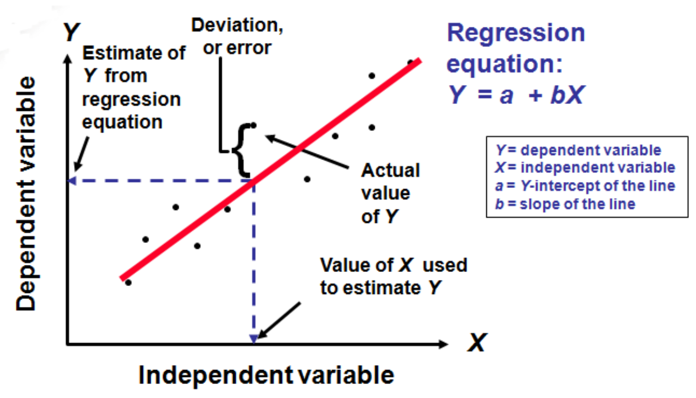


Figure 5.2: Linear Regression

5.3.2 ARIMA

ARIMA stands for Autoregressive Integrated Moving Average models. ARIMA is a time series forecasting technique, that aims to describe the autocorrelations in the data. This section first give some basic definitions and tools that are needed to define and understand ARIMA models. Backshift operator is handy when working with time series lags. Stationarity, which is a basic requirement for the input time series for ARIMA modeling, is described, along with differencing, the tool to convert non-stationary time series into stationary ones. After describing the AR (Autoregressive), and MA (Moving average) models, the ARIMA model, which is the combination of these, is introduced.

There are two basic operators that are essential to explain the AR, MA, ARIMA models. The backshift operator to describe time shifts in time series, and difference operator to build *differenced series* that contains the change between consecutive observations in the original series.

Backshift operator. The backward shift operator B is a useful notational device when working with time series lags:

Definition 1

$$B^k x_t = x_{t-k}$$

where k is the time shift or lag.

Difference operator. The backward difference operator is another important operator, defined as:

Definition 2

$$\nabla x_t = x_t - x_{t-1} = (1 - B)x_t.$$

Stationarity and differencing. Usually time series are based upon time-dependent phenomena with many unknown factors. To derive a stochastic time series model, there has to be some regularity in the time series behavior. Loosely speaking, a time series $X_t, t = 0, \pm 1, ...$ is said to be stationary if it has statistical properties similar to those of the time shifted series $X_{t+h}, t = 0, \pm 1, ...$, for each integer h. Restricting attention to those properties that depend only on the first- and second order moments of X_t.

Time series whose statistical properties such as mean, variance, autocorrelation, etc. are all constant over time are stationary. A stationary series is relatively easy to predict: one simply predicts that it's statistical properties will be the same in the future as they have been in the past. The same stochastic process that describes the observed time series can be used to generate next time series elements (forecasts).

Although, many of the time series have trend and/or seasonal components and so they are not stationary. Using mathematical transformations like differencing (see above) may transform non-stationary time series into a stationary one. Differencing creates a new series that contains the change between consecutive observations in the original series. One can apply differencing on a differentiated time series too if it is still not stationary, this is called the second order differentiated time series.

Other mathematical transformation is the seasonal differentiating. A seasonal difference is the difference between an observation and the corresponding observation from the previous season [61] [13] [68]. An example of the differentiation is shown in the Figure B.1 in the Appendix.

Autoregression. In an autoregression model, forecasting the variable of interest using a linear combination of past values of the variable is done. The autoregressive model specifies that the output variable depends linearly on its own previous values [61] [20]. Thus an autoregressive model of order p can be written as

$$x_t = c + \phi_1 x_{t-1} + \phi_2 x_{t-2} + \ldots + \phi_p x_{t-p} + e_t,$$

where c is a constant, ϕ_1,ϕ_2,ϕ_3 are the autoregressive model parameters and e_t is a random error component. This is like a multiple regression but with lagged values of x_t as predictors. One can refer to this as an AR(p) model. Put into words, each observation is made up of a random error component e_t and a linear combination of prior observations.

Moving Average. Rather than using past values of the forecast variables in a regression, the moving average model uses past forecast errors in a regression-like model [61] [20].

$$x_t = c + e_t + \theta_1 e_{t-1} + \theta_2 e_{t-2} + \ldots + \theta_q e_{t-q},$$

where c is a constant, θ_1,θ_2,θ_3, are the moving average model parameters. e_t is forecast errors. We refer to this as an MA(q) model. Put into words, each observation is made up of a random error component and a linear combination of prior random errors.

ARIMA model. If we combine differencing with autoregression and the moving average model, we obtain a non-seasonal ARIMA(p,d,q) model. where p is the order of autoregression, d is the order of differenciating, and q is the order of the moving average model. ARIMA is an acronym for AutoRegressive Integrated Moving Average model ("integration" in this context is the reverse of differencing). The full model can be written as

$$x'_t = c + \phi_1 x'_{t-1} + \cdots + \phi_p x'_{t-p} + \theta_1 e_{t-1} + \cdots + \theta_q e_{t-q} + e_t,$$

Where $x't$ is the differenced series (it may have been differenced more than once). The „predictors“ on the right hand side include both lagged values of y_t and lagged errors. This is called an ARIMA(p,d,q) model. Where p is the order of autoregression ($AR(p)$), d is the order of differencing, and q is the order of moving average ($MA(q)$).

The selection of the order (p,d,q) and the estimation of the parameters ($c, \phi_1, \ldots, \phi_p, \theta_1, \ldots, \theta_q$) is typically done by calculating and testing the forecast on a subset of the input data. Usually the model is chosen with a minimal number of parameters and the least forecast errors.

5.3.3 Simple Exponential Smoothing

This method is suitable for forecasting data with no trend or seasonal pattern but clearly the data can change over time. Forecasts are calculated using weighted averages where the weights decrease exponentially as observations come from further in the past. The smallest weights are associated with the oldest observations:

$$\hat{x}_{T+1|T} = \alpha x_T + \alpha(1-\alpha)x_{T-1} + \alpha(1-\alpha)^2 x_{T-2} + \cdots,$$

where $0 \leq \alpha \leq y_1$ is the smoothing parameter. The one-step-ahead forecast for time $T+1$ is a weighted average of all the observations in the series $y_1, \ldots, y_T$. The rate at which the weights decrease is controlled by the parameter α.

5.3.4 Holt's linear trend method

This method involves a forecast equation and two smoothing equations (one for the level and one for the trend). This is an extension of exponential smoothing to take into account a possible linear trend:

Forecast equation:	$\hat{x}_{t+h} t = \ell_t + hb_t$
Level equation:	$\ell_t = \alpha y_t + (1-\alpha)(\ell_{t-1} + b_{t-1})$
Trend equation:	$b_t = \beta^*(\ell_t - \ell_{t-1}) + (1-\beta^*)b_{t-1}$

where ℓ_t denotes an estimate of the level of the series at time t, b_t denotes an estimate of the trend (slope) of the series at time t, α is the smoothing parameter for the level, $0 \leq \alpha \leq 1$ and β is the smoothing parameter for the trend, $0 \leq \beta^* \leq 1$

5.4 Seasonal Forecasting

This section presents methods that are built mostly upon existing linear trend estimation methods described in the previous chapter. The observed patterns are without exception strongly seasonal, therefore in a real case scenario seasonal forecasting methods will more likely to outperform trend estimation methods. Therefore this thesis concentrates on these methods. Seasonality means periodic fluctuations, this can be detected with various methods:

- ACF/PCF plots: Will show significant spike/ exponential decay at the lag of the seasonal period
- The power spectrum (or periodogram) plots the power versus frequency. Cyclic (that is, repetitive or seasonal patterns) will show up as large spikes located at their frequencies.The power spectrum is the discrete Fourier transform of the autocovariance function of an appropriately smoothed version of the original series.
- Seasonal Component after STL Decomposition

5.4.1 Naïve Seasonal Forecasting

The most simple type of seasonal forecast would be just fixing the seasonal period (e.g. for one day) and for the forecast period the same values from the previous period are repeated:

$$\hat{x}_t = x_{t-h}$$

where h is the number of values in the seasonal period.

Another simple method is based on the STL decomposition earlier. One can take the seasonal component after STL decomposition and repeat it over and over again as the forecast for the next period.

5.4.2 Seasonal ARIMA

Earlier it was described how ARIMA can be used for trend estimation, and it can be also used for modeling seasonal data. To include seasonal terms in a model it is necessary to multiplicative combine an ordinary non-seasonal ARIMA model with an ARIMA model that is extended to a seasonal period m. Therefore the AR and the MA model are extended to the lags of the seasonal period m. A seasonal ARIMA model is formed by including additional seasonal terms:

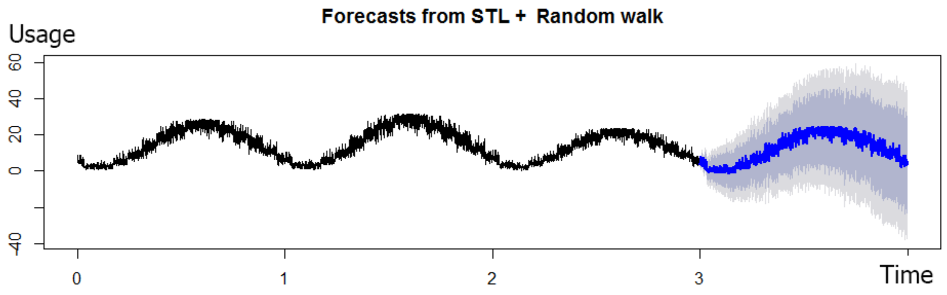


Figure 5.3: Extrapolating seasonal component after STL decomposition in R: Black is the input data, Blue is the forecast. The testdata contains 3 days of CPU usage (4320 data point), forecasted period is 1 day (1440 data point).

$$ARIMA(p,d,q)(P,D,Q)_m$$

where m is the number of periods per season. Uppercase notation is used for the seasonal parts of the model, and lowercase notation for the non-seasonal parts of the model. The seasonal part of the model consists of terms that are very similar to the non-seasonal components of the model, but they involve backshifts of the seasonal period. The additional seasonal terms are simply multiplied with the non-seasonal terms [61].

For example $ARIMA(1,1,1)(1,1,1)_{24}$ model is for hourly data ($m = 24$) and can be written as:

$$\underbrace{(1-\phi_1 B)}_{Non-Seasonal AR(1)} \underbrace{(1-\phi_1 B^{24})}_{Seasonal AR(1)} \underbrace{(1-B)}_{NonSeasonal Difference} \underbrace{(1-B^{24})x_t}_{Seasonal Difference} = \underbrace{(1+\Theta_1 B)}_{Non-Seasonal MA(1)} \underbrace{(1+\Theta_1 B^{24})e_t}_{Seasonal MA(1)}$$

automatic order selection and parameter estimation implemented in R (auto.arima). It is done by calculating and testing the forecast on a subset of the input data. Usually the model is chosen with minimal number of parameters and the least forecast errors.

5.4.3 Seasonal Exponential Smoothing

Holt's linear method has been extended to capture seasonality. The Holt-Winters seasonal method comprises the forecast equation and three smoothing equations — one for the level ℓ_t, one for trend b_t, and one for the seasonal component denoted by s_t, with smoothing parameters α, β^* γ. m is used to denote the period of the seasonality. (e.g. $m = 24$ for hourly data)

The mathematical description of the additive Holt-Winter's method is given by:

Forecast	$\hat{x}_{t+h} = \ell_t + hb_t + s_{t-m+h}$
Level	$\ell_t = \alpha(x_t - s_{t-m}) + (1-\alpha)(\ell_{t-1} + b_{t-1})$
Trend	$b_t = \beta^*(\ell_t - \ell_{t-1}) + (1-\beta^*)b_{t-1}$
Seasonality	$s_t = \gamma(x_t - \ell_{t-1} - b_{t-1}) + (1-\gamma)s_{t-m}$

The level equation shows a weighted average between the seasonally adjusted observation $(x_t - s_{t-m})$ and the non-seasonal forecast $(\ell_{t-1} + b_{t-1})$for time t. The trend equation is identical to Holt's linear method. The seasonal equation shows a weighted average between the current seasonal index, $(x_t - \ell_{t-1} - b_{t-1})$, and the seasonal index of the same season last period. Depending how the components are calculated (None, Additive, Additive damped, Multiplicative, Multiplicative damped) there exists all together 15 variation of exponential smoothing [39].

		Seasonal Component	
Trend Component	N	A	M
None	(N,N)	(N,A)	(N,M)
Additive	(A,N)	(A,A)	(A,M)
Additive Damped	(Ad,N)	(Ad,A)	(Ad,M)
Multiplicative	(M,N)	(M,A)	(M,M)
Multiplicative Damped	(Md,N)	(Md,A)	(Md,M)

Table 5.1: ETS models

For each models of the Exponential Smoothing (ETS) method above, there are two possible innovation state space models, one corresponding to a model with additive errors and the other to a model with multiplicative errors. Thus there are 30 potential models [39].

For the model selection one needs to define an optimization objective. In R forecast package both for ARIMA order selection, and for ETS parameter estimation the optimization objective is given by Akaike's Information Criterion

$$AIC = N \log\left(\frac{SSE}{N}\right) + 2(k+2),$$

The selection algorithm keeps altering the model and selects the one with the lowest AIC. Alternatively minimizing only the Sum of Squared Errors (SSE) could be a reasonable objective too:

$$SSE = \sum_{i=1}^{N} e_i^2.$$

where N is the number of forecast points, and e_i is the difference between the forecasted and the actual value in the time i.

5.4.4 Fourier Forecast

Thanks to its capability in terms of decomposing a function into a sum of sinusoids of different frequencies, amplitude and phase, Fourier analysis can be used effectively for seasonal forecasting. Fourier transform takes a time series and maps it into a frequency spectrum in the frequency domain. The discrete version of the Fourier transform can be quickly calculated using fast Fourier transform (FFT) algorithms. The FFT algorithm significantly reduces computational times compared to the standard Fourier transform, and specifically from $O(n^2)$ to $(nlog(n))$ this

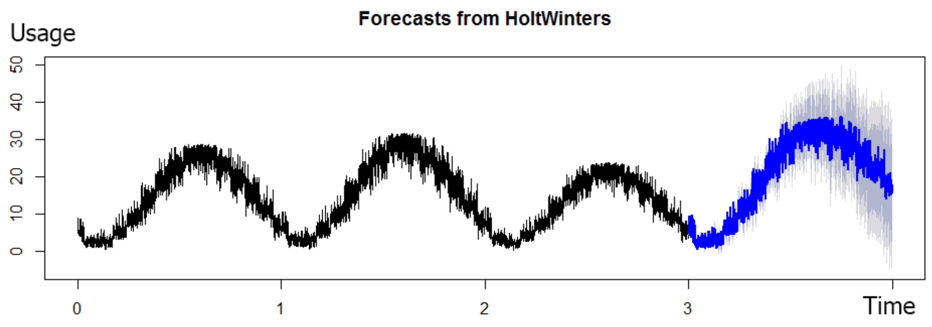


Figure 5.4: Using Holt Winter's method in R: Black is the input data, Blue is the forecast. The testdata contains 3 days of CPU usage (4320 data point), forecasted period is 1 day (1440 data point).

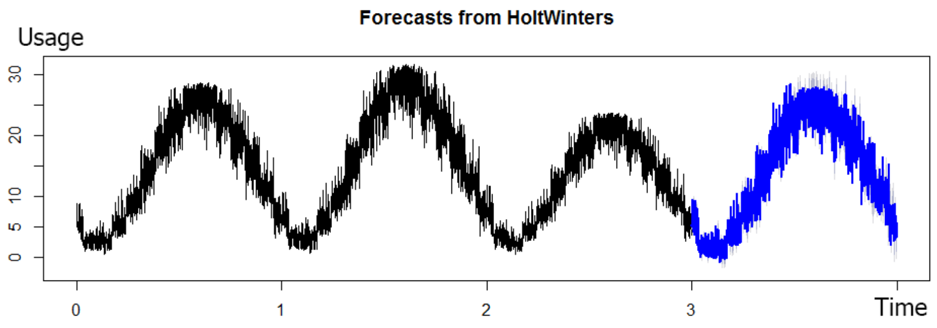


Figure 5.5: Using Holt Winter's method with suppressed trend forecast in R: Black is the input data, Blue is the forecast. The testdata contains 3 days of CPU usage (4320 data point), forecasted period is 1 day (1440 data point).

makes this forecast method the fastest. The FFT returns complex numbers from which it is possible to extract information – frequency (f), amplitude (A), and phase (φ) – of $N/2$ periodic waves that form the sales pattern, where N is the total number of time periods of the data, which is also the size of the sample. The result of the Fourier transform is:

$$x_t = \sum_{i=1}^{k}(\alpha_i sin 2\pi f_i t + b_i cos 2\pi f_i t)$$

One can select the optimal number of fourier terms by optimizing AIC or SSE, similarly as before [4].

5.4.5 Neural Network Autoregression

A neural network can be thought of as a network of „neurons“ organized in layers. The predictors (or inputs) form the bottom layer, and the forecasts (or outputs) form the top layer. There may be

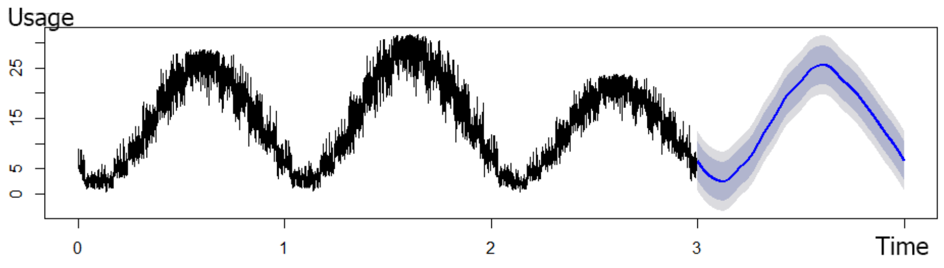


Figure 5.6: Using Fourier forecast in R with 10 fourier terms: Black is the input data, Blue is the forecast. Grey area is the confidence level. The testdata contains 3 days of CPU usage (4320 data point), forecasted period is one day (1440 data point).

intermediate layers containing „hidden neurons“. Sample neural network is illustrated in Figure 5.7.

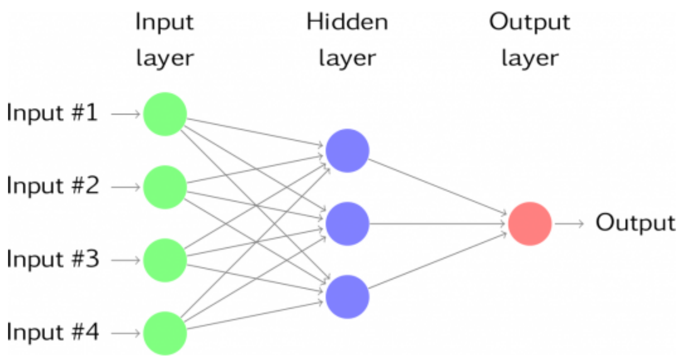


Figure 5.7: Network with Hidden layer [61]

This is known as a multilayer feed-forward network where each layer of nodes receives inputs from the previous layers. The outputs of nodes in one layer are inputs to the next layer. The inputs to each node are combined using a weighted linear combination. The result is then modified by a nonlinear function before being output. For example, the inputs into hidden neuron j in Figure above are linearly combined to give

$$z_j = b_j + \sum_{i=1}^{4} w_{ij} x_i$$

where z_j is the output of neuron j, w_{ji} is the weight on the connection from neuron i to j, x_i is the input signal of the neuron i and b_j is the bias of neuron j.

In the hidden layer, this is then modified using a nonlinear function such as a sigmoid,

$$s(z) = \frac{1}{1 + e^{-z}}$$

that gives the input for the next layer. This tends to reduce the effect of extreme input values, thus making the network somewhat robust to outliers. The parameters b_1, b_2, b_3 and $w_{1,1}, \dots, w_{4,3}$ are „learned“ from the data. The values of the weights are often restricted to prevent them becoming too large. The parameter that restricts the weights is known as the „decay parameter“ and is often set to be equal to 0.1.

In R forecast package, **nnetar()** function is implemented that use lagged values in a neural network autoregression. It consider feed-forward networks with one hidden layer, the notation NNAR(p,k) to indicate there are p lagged inputs and k nodes in the hidden layer. For example, a NNAR(9,5) model is a neural network with the last nine observations $(x_{t-1}, x_{t-2}, \dots, x_{t-9})$ used as inputs to forecast the output y_t, and with five neurons in the hidden layer. With seasonal data, it is useful to also add the last observed values from the same season as inputs. We note such model as $NNAR(p, P, k)_m$, which has as input: $(x_{t-1}, x_{t-2}, \dots, x_{t-p}, x_{t-m}, x_{t-2m}, x_t - P_m)$ and k neurons in the hidden layer.

A $NNAR(p, P, 0)_m$ model is equivalent to an $ARIMA(p, 0, 0)(P, 0, 0)_m$ model but without the restrictions on the parameters to ensure stationarity and with nonlinear functions. Since ARIMA function fails in R on high dimensional data (above dimensionality of 350 or higher) we will use nnetar function in our simulation [61].

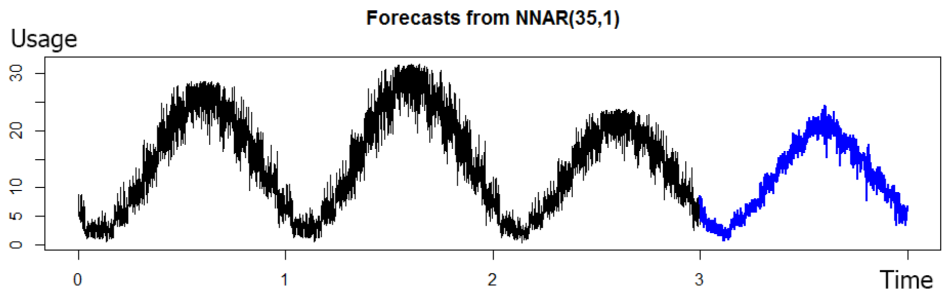


Figure 5.8: Using NNAR method in R: Black is the input data, Blue is the forecast. The testdata contains 3 days of CPU usage (4320 data point), forecasted period is 1 day (1440 data point).

5.5 Complex Seasonality Forecasting

5.5.1 TBATS

Using Holt-Winters Seasonal method it is possible to capture the daily seasonality of the data, however over the daily seasonality we have a hourly seasonality in the data. To capture this complex multi level seasonality we use the TBATS stochastic process modeling framework described in [21]. TBATS is an acronym for: Trigonometric, Box–Cox transformation, ARMA errors, Trend, and Seasonal components. The seasonal component of the TBATS model is based on Fourier Series. Box-Cox transformation is used to stabilize the variation in the data. The advantage of this model besides the multi seasonality that it can be comfortably applied for high frequency (per minute) data.

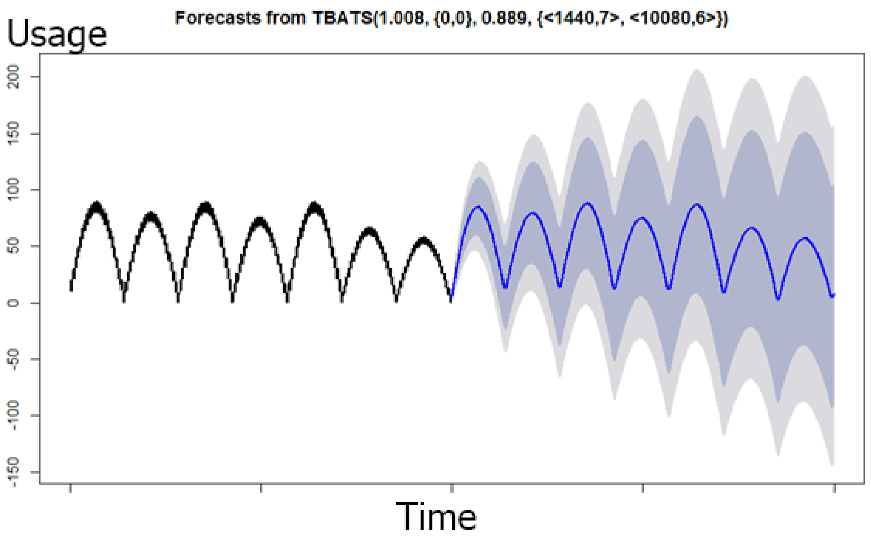


Figure 5.9: Using tBATS method in R: Black is the input data, Blue is the forecast. The testdata contains 7 days of CPU usage (10080 data point), forecasted period is one week (10080 data point). Data contains double seasonality: weekly and daily.

5.5.2 Double Seasonal Holt-Winters

[69] extended the Holt Winters exponential smoothing to handle two seasonal periods (e.g. Weekly and daily seasonality). Double-seasonal Holt-Winters method uses additive trend and multiplicative seasonality, where there are two seasonal components which are multiplied together. D_t and W_t represent the daily and weekly seasonal factors. Let S_t be the local level, and T_t be the local trend. Let α, γ, δ, ω denote the smoothing constants. Then, the updating equations for the multiplicative Holt-Winters Double Seasonal Model are given in equations:

Level	$\mathrm{S}_t = \alpha \left(\frac{X_t}{D_{t-s_1} W_{t-s_2}} \right) + (1-\alpha)(S_{t-1} + T_{t-1})$
Trend	$\mathrm{T}_t = \gamma(S_t - S_{t-1}) + (1-\gamma)T_{t-1}$
First level seasonality	$\mathrm{D}_t = \delta \left(\frac{X_t}{S_t W_{t-s_2}} \right) + (1-\delta)D_{t-s_1}$
Second level seasonality	$\mathrm{W}_t = \omega \left(\frac{X_t}{S_t D_{t-s_1}} \right) + (1-\omega)W_{t-s_2}$
Forecast	$\mathrm{X}_{t+k} = (S_t + kT_t)D_{t-s_1+k}W_{t-s_2+k}$

The forecast is given by a combined equation of trend, level, first and second level seasonality equations. In the equation s_1 is the period of the first level seasonality (e.g. daily) and s_2 is the second level seasonality (e.g. weekly). The smoothing parameters α, γ, δ, ω are controlling the impact of each sub-equation. Note if all smoothing parameter are set to 1 that gives a naive forecast in which the next value only depends on the latest observation. The smoothing hyperparameters α, γ, δ, ω are optimized using the minimum one step ahead mean squared error criterion. Optimal values of the smoothing hyperparameters can be obtained through a genetic algorithm search procedure as in [69].

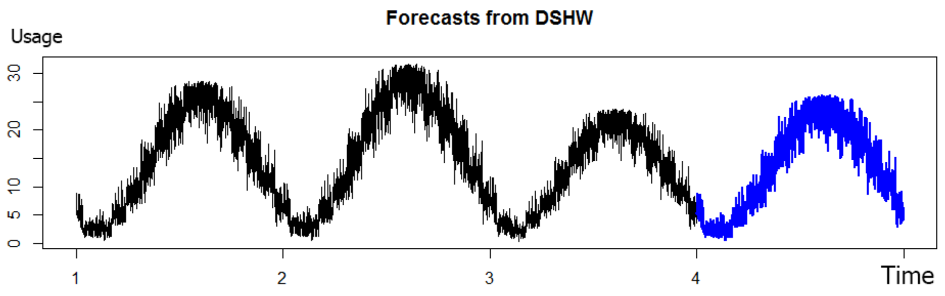


Figure 5.10: Using Double Seasonal Holt Winter's method in R: Black is the input data, Blue is the forecast. The test data contains 3 days of CPU usage (4320 data point), forecasted period is one day (1440 data point). Data does not contain double seasonality!

5.6 Other Methods

5.6.1 Markov Chain Model

Markov chains are an other class of stochastic process to describe time series. The main idea behind markov chains that based on the historical data it construct finite number of so called markov states and based on historical changes in the data it „learns" the transition probability between the states. This probability distribution will be the basis of predicting the future steps of the data. More formally a Markov chain, studied at the discrete time points $0, 1, 2,$, is characterized by a set of states $S = s_1, s_2...s_n$ and the transition probabilities p_{ij} between the states. Here, p_{ij} is the probability that the Markov chain is at the next time point in state j, given that it is at the present time point at state i. The matrix P with elements p_{ij} is called the transition probability matrix of the Markov chain.

PRESS [34] uses markov chains to detect application intern state changes and use it to predict future resource usage. The author in the paper discretizes usage values into $M = 40$ states and builds a transaction probability matrix P which is then an $M \times M$ matrix where the element p_{ij} denotes the probability of the state change from i to j. The model predicts the state at a future time by determining the most probable future state. An example state-driven prediction model is shown on 5.11. In this example memory usage ranges from 0 to 30, and the range is partitioned into 3 states. The arcs are labeled with their transition probability.

The further time t goes in the future, the weaker the correlation between the model and the actual demand. This means that markov chain based forecasting is only good for short term [34].

5.6.2 Wavelet based demand prediction

In an other related article wavelet decomposition based forecasting method is introduced, that is used for forecasting demand for up to 2 minutes ahead. The system is called AGILE. AGILE

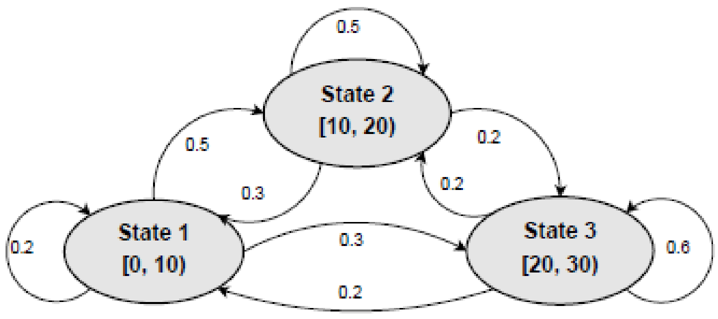


Figure 5.11: Example markov chain model [34]

provides online resource demand prediction using a sliding window D (e.g., D = 6000 seconds) of recent resource usage data. It employs wavelet transforms to make predictions: at each sampling instant t, predicting the resource demand over the prediction window of length W (e.g., W = 120 seconds). The basic idea is to first decompose the original resource demand time series into a set of wavelet based signals. Than, predictions are performed for each decomposed signal separately. Finally, the future resource demand is synthesized by adding up all the individual signal predictions. Figure 5.12 illustrates our wavelet based prediction results for an Apache web server's CPU demand trace. Wavelet transforms have two key configuration parameters: 1) the wavelet function to use, and 2) the number of scales. AGILE dynamically configures these two parameters in order to minimize the prediction error. For each sliding window D, AGILE selects the wavelet function that results in the smallest Euclidean distance between the approximation signal and the original signal.

The method is further described in [54] and based on the results it outperforms existing prediction methods for the prediction horizon of 2 minutes. We consider this approach as a short term forecasting method [54].

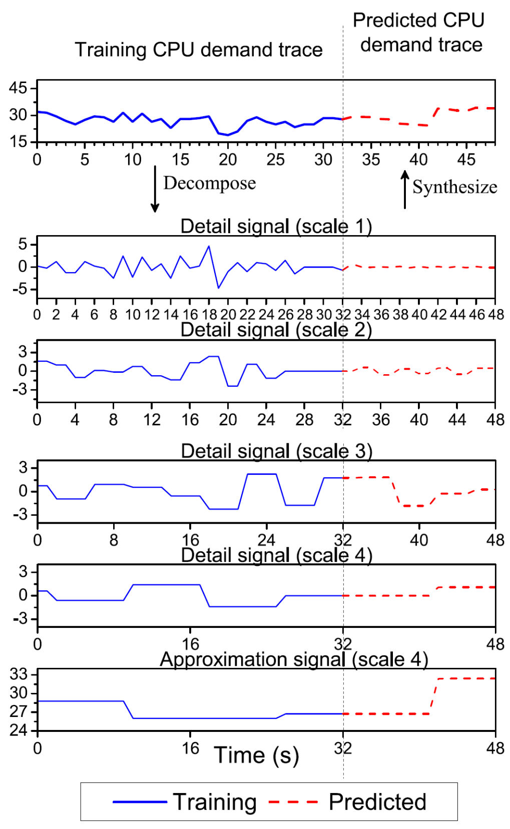


Figure 5.12: Example wavelet decomposition and prediction of an Apache web server CPU demand [54]

Wavelet transforms decompose a signal into a set of wavelets at increasing scales. Wavelets at higher scales have larger duration, representing the original signal at coarser granularities. Each scale i corresponds to a wavelet duration of L_i seconds, typically $L_i = 2_i$. For example, in Figure 5.12, each wavelet at scale 1 covers 21 seconds while each wavelet at scale 4 covers $24 = 16$ seconds. After removing all the lower scale signals called detailed signals from the original signal, we obtain a smoothed version of the original signal called the approximation signal. For example, in Figure 5.12, the original CPU demand signal is decomposed into four detailed signals from scale 1 to 4, and one approximation signal. Then the prediction of the original signal is synthesized by adding up the predictions of these decomposed signals.

5.7 Sporadic forecasting

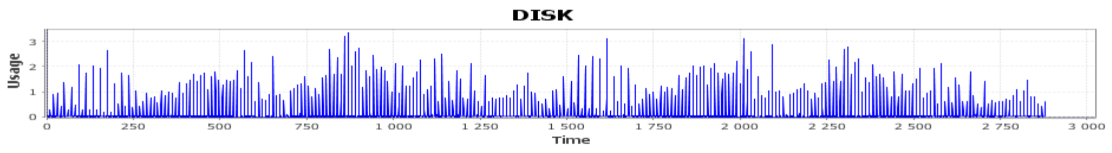


Figure 5.13: Observed Disk IO traces

Based on the observed traces (figure 5.13) it can be concluded that during the workload generation process the hard drive was not used in every time: most of the times the load is near zero and other times it is spiking up. This is called sporadic trace and it utilizes a different method for predicting future values. (Averaging input before forecast leads to significant forecast errors.) The Sporadic forecast process is presented on figure 5.14.

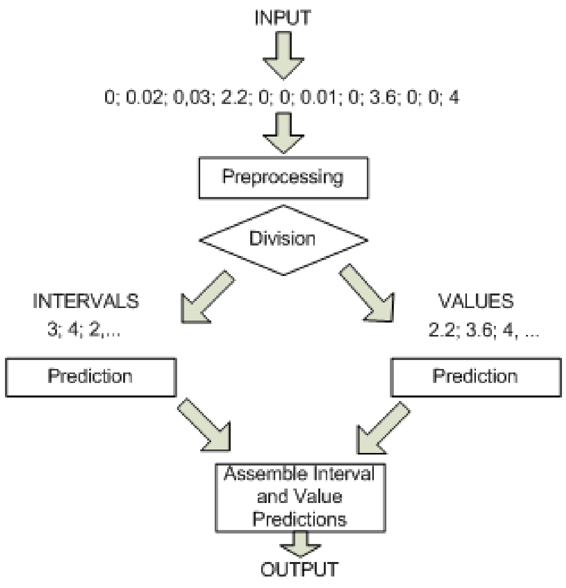


Figure 5.14: Sporadic Forecasting Process

First there is some preprocessing needed because the low values are usually not zero just near to it: values lower than a threshold (<0.05) will be set to zero. Then the input time series is separated into two series: the first contains the intervals between the non-zero values, the second is the non zero values from the original time series. For both time series one of the previously described forecasting methods is used. Finally the predicted values and the predicted intervals between the values will be assembled as one forecast time series. In this case, fourier based forecasting was used for both time series. The result is shown on figure 5.15.

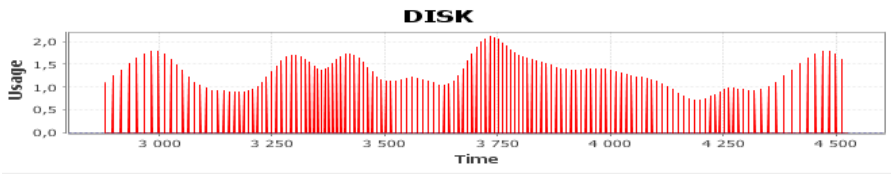


Figure 5.15: Forecasted Disk IO traces

Sporadic forecast is a very interesting problem, with sporadic-fourier forecast one still has a higher error rate for predicting Disk I/O usage than predicting all the other metrics.

5.8 Evaluation forecast performance

In order to estimate model parameters and compare different models one needs to evaluate the performance of the different approaches. These are also known as performance metrics. Each of these measures is a function of the actual and forecasted values of the time series. In this section we describe the commonly used performance measures and their important properties. In each of the forthcoming definitions, y_t is the actual value, f_t is the forecasted value, $e_t = y_t - f_t$ is the forecast error and n is the size of the test set. Also, $\hat{y} = \frac{1}{n}\sum_{t=1}^{n} y_t$ is the test mean and $\sigma^2 = \frac{1}{n-1}\sum_{t=1}^{n}(y_t - \hat{y})^2$ is the test variance [1] [40] [53].

This section provides a list of the important performance evaluation metrics that helps one to compare the different approachs and determine which approach is superior for a given forecast scenario. The properties of each metric are also described.

- **Mean Forecast Error (ME)**

  Mean forecast error (ME) is a measure of the average deviation of forecasted values from actual ones. It shows the direction of error and thus is also termed as the *Forecast Bias*. In ME, the effects of positive and negative errors cancel out and there is no way to know their exact amount. A zero ME does not imply that forecasts are perfect, i.e. contain no error; rather it only indicates that forecasts are on proper target. ME does not penalize extreme errors. It depends on the scale of measurement and also affected by data transformations. For a good forecast, i.e. to have a minimum bias, it is desirable that the ME is as close to zero as possible. It is defined as follows [40]:

$$ME = \frac{1}{n}\sum_{t=1}^{n} e_t$$

- **The Mean Absolute Error (MAE)**

  The Mean Absolute Error (MAE) measures the average absolute deviation of forecasted values from original ones. It is also termed as the Mean Absolute Deviation (MAD). It shows the magnitude of overall error, occurred due to forecasting. In MAE, in contrasted with ME, the effects of positive and negative errors do not cancel out. Unlike ME, MAE does not provide any information about the direction of errors. For a good forecast, the obtained MAE should be as small as possible. Like ME, MAE is also scale-dependent measure. Extreme forecast errors are not penalized by MAE. MAE is defined as follows [40]:

$$MAE = \frac{1}{n}\sum_{t=1}^{n} |e_t|$$

- **The Mean Absolute Percentage Error (MAPE)**

  The Mean Absolute Percentage Error (MAPE) represents the percentage of average absolute error occurred. It is independent of the scale of measurement, but affected by data transformation. It does not show the direction of error. MAPE does not penalize extreme deviations. In this measure, opposite signed errors do not cancel each other out. MAPE is defined as follows [40]:

$$MAPE = \frac{1}{n}\sum_{t=1}^{n} \left|\frac{e_t}{y_t}\right| \times 100$$

- **The Mean Percentage Error (MPE)**

  The Mean Percentage Error (MPE) represents the percentage of average error occurred. It has similar properties as MAPE, except, it shows the direction of error occurred and opposite signed errors do cancel each other out. Like ME, by obtaining a value of MPE close to zero, we can not conclude whether the corresponding model performed well. It is desirable that for a good forecast the obtained MPE should be small. MPE is defined as follows [40]:

$$MPE = \frac{1}{n}\sum_{t=1}^{n} \left(\frac{e_t}{y_t}\right) \times 100$$

- **The Root Mean Squared Error (RMSE)**

  The Root Mean Squared Error (RMSE) is the square root of average squared deviation of forecasted values. As here the opposite signed errors do not offset one another, RMSE gives an overall idea of the error occurred during forecasting. It penalizes extreme errors occurred while forecasting. RMSE emphasizes the fact that the total forecast error is in fact much affected by large individual errors, i.e. large errors are much expensive than small errors. RMSE does not provide any information about the direction of overall error. RMSE is sensitive to the change of scale and data transformations. Although RMSE is a good measure of overall forecast error, but it is not as intuitive and easily interpretable as the other measures discussed before. RMSE is defined as Follows [40]:

$$RMSE = \sqrt{\frac{1}{n}\sum_{t=1}^{n} e_t^2}$$

- **Mean absolute scaled error (MASE)**

  Mean absolute scaled error (MASE) is less than one if it arises from a better forecast than the average one-step, naïve forecast computed in-sample. Conversely, it is greater than one if the forecast is worse than the average one-step, naïve forecast computed in-sample This scale-free error metric can be used to compare forecast methods on a single series and also to compare forecast accuracy between series. MASE never gives infinite or undefined values (except when historical data are all equal which makes no sense) [40].

  A scaled error is defined as q_t:

$$q_t = \frac{|e_t|}{\frac{1}{n-1}\sum_{i=2}^{n} |Y_i - Y_{i-1}|}$$

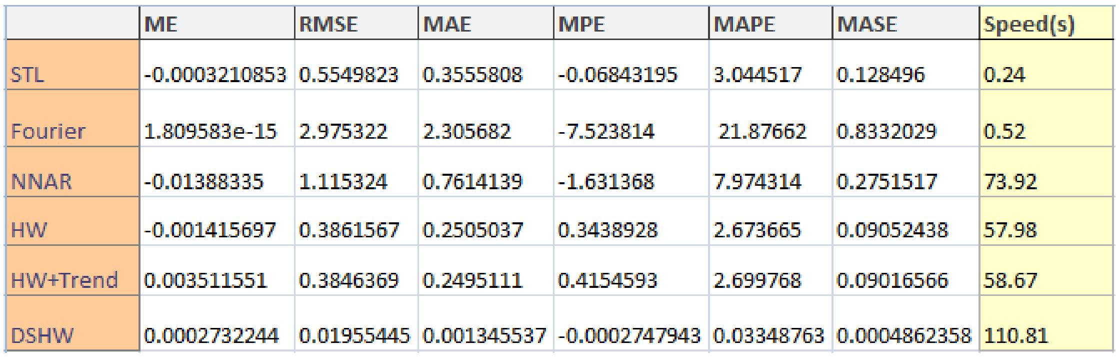
$$MASE = mean(|q_t|)$$

5.9 Summary

Evaluating seasonal forecasting methods with the described accuracy metrics on the generated 3 days of CPU resource usage data:

Note that results in 5.2 are produced by measuring the „goodness“ of the fit. This means that the accuracy measures of the forecasts are based on the error between the points of the original time series and the points of the fitted time series.

The following table summarizes the finding of the strengths and weaknesses of each of the introduced forecasting methods. The optimal forecasting horizon that the method can be applied onto is described, along with the computational overhead that the calculation of the forecasts costs.



	ME	RMSE	MAE	MPE	MAPE	MASE	Speed(s)
STL	-0.0003210853	0.5549823	0.3555808	-0.06843195	3.044517	0.128496	0.24
Fourier	1.809583e-15	2.975322	2.305682	-7.523814	21.87662	0.8332029	0.52
NNAR	-0.01388335	1.115324	0.7614139	-1.631368	7.974314	0.2751517	73.92
HW	-0.001415697	0.3861567	0.2505037	0.3438928	2.673665	0.09052438	57.98
HW+Trend	0.003511551	0.3846369	0.2495111	0.4154593	2.699768	0.09016566	58.67
DSHW	0.0002732244	0.01955445	0.001345537	-0.0002747943	0.03348763	0.0004862358	110.81

Table 5.2: Forecast Accuracy Metrics on 3 days cpu testdata

#	ME	RMSE	MAE	MPE	MAPE	MASE
1.	DSHW	DSHW	DSHW	DSHW	DSHW	DSHW
2.	STL	HW+Trend	HW+Trend	STL	HW	HW+Trend
3.	HW	HW	HW	HW	HW+Trend	HW
4.	HW+Trend	STL	STL	HW+Trend	STL	STL
5.	NNAR	NNAR	NNAR	NNAR	NNAR	NNAR
6.	Fourier	Fourier	Fourier	Fourier	Fourier	Fourier

Table 5.3: Seasonal forecasting methods ranked by forecast accuracy. Different accuracy metrics are used as the base of the comparison.

Method	Disadvantages	Advantages	Optimal Horizon	Computational Overhead
ARIMA (Non-seasonal)	Fails on high dimensional data, expensive internal order selection	Flexible modeling for middle term trend and noise	Mid	Moderate
ARIMA (Seasonal)	Fails on high dimensional data, expensive internal order selection	Flexible modeling for long term season and noise	Long	Moderate
ETS (Non-seasonal)	Expensive Model Selection	Flexible modeling for middle term level, trend and noise	Mid	Moderate
ETS (Seasonal)	Expensive Model Selection	Flexible modeling for seasonal, level, trend and noise components	Long	Moderate
NNAR	Similar to seasonal ARIMA, but more expensive	Good for predicting very similar seasonal cycles	Long	Moderate
Fourier	Bad at predicting short term fluctuations	Fast approximation of seasonal time series	Long	Low
STL	Works bad when seasonal cycles are different	Fast and good approximation when seasonal cycles are steady and the amount of short term fluctuation is steady	Long	Low
Markov Chains	Can not predict for long term	Good at predicting short term fluctuation	Short	N/D
Wavelet based	Highly dependent on configuration of wavelet functions	Has good results for predicting for short-mid term	Short-Mid	N/D
DSHW	Expensive calculation, possible overfitting when used for simple seasonal data	Applicable when time series have two level seasonality	Long	High
TBATS	Expensive, R implementation needs to be adjusted to work on hourly/daily seasonal cycles	Very complex method that is applicable for any kind of seasonal behavior (non-integer, multi-level seasonality etc.)	Long	High

Table 5.4: Summary of forecasting methods described in this chapter

CHAPTER 6

Model Of The Dynamic Forecast Selection

The previous chapter described how the various forecasting methods work and what their strengths and weaknesses are. In this chapter a dynamic and automatic selection procedure to find the best forecasting strategy (seasonal forecasting, trend interpolation, short term prediction), forecasting method, and forecasting method parameters, for any input time series, is introduced. During the monitoring process different kinds of time series may be observed. Finding the optimal forecasting method is a challenging and computationally expensive task. Finding the best performing forecasting method leads to more accurate forecasts, which directly results in more accurate SLA detection and violation prevention in cloud computing systems.

This chapter begins by explaining the model selection and parameter optimization process applied for ARIMA and ETS models in the R forecast package [62]. Then these selection processes are extended to be applicable to any forecasting tools. Since this process is a computationally expensive task, this chapter also describes dimensionality reduction techniques that have the potential to ease the forecast selection process.

6.1 The Model Selection Problem

One of the most difficult tasks in time series analysis is model selection—the choice of an adequate model for a given set of data. Even trained time series analysts will often find it difficult to choose a model. George Box famously stated that “All models are wrong but some are useful” ([12], p. 424). The models that are fitted to time series data are selected on the basis of a study of the properties of the data, in particular, sample ACF and PACF. The problem is finding or selecting a model that describes underlying mechanisms or phenomena behind the time series. If the exact mechanism is fully known and understood then providing an exact mathematical expression or theoretical model would be not a problem. In the case of this thesis the underlying mechanism that induces the resource usage (e.g. number of users) is not fully known and cannot be described by exact mathematical expressions. Therefore, the historical time series data (observations) is used to derive a model for forecasting and optimizing model parameters.

There are a lot of catches during the process though: for example forecasting with a polynomial model that fits perfectly to the input time series, is more likely to deliver bad predictions (overfitting), and so is a highly averaged model.

The problem is how to automatically decide which model is adequate. This chapter introduces the model selection process used by ARIMA and ETS methods. These iterative model building and parameter optimization processes are extended so that they can be used with other previously described forecasting methods. Furthermore, this chapter is dedicated to describing suggested model selection processes, and diagnostic tools to evaluate them.

6.1.1 ARIMA Model selection

In order to obtain the optimal parameters for $ARIMA(p, d, q)(P, D, Q)_m$ while forecasting on arbitrary univariante input time series, (e.g. a set of observations about some resource usage of a cloud application) auto.arima method in R forecast package implements the following algorithm [39]:

Step 1:

We try four possible models to start with.

- $ARIMA(2, d, 2)$ if $m = 1$ and $ARIMA(2, d, 2)(1, D, 1)$ if $m > 1$.
- $ARIMA(0, d, 0)$ if $m = 1$ and $ARIMA(0, d, 0)(0, D, 0)$ if $m > 1$.
- $ARIMA(1, d, 0)$ if $m = 1$ and $ARIMA(1, d, 0)(1, D, 0)$ if $m > 1$.
- $ARIMA(0, d, 1)$ if $m = 1$ and $ARIMA(0, d, 1)(0, D, 1)$ if $m > 1$.

Of these four models, the one with the smallest AIC value is selected. This is called the „current“ model and is denoted by $ARIMA(p, d, q)$ if $m = 1$ or $ARIMA(p, d, q)(P, D, Q)_m$ if $m > 1$

Step 2:

Up to thirteen variations on the current model are considered:

- where one of p, q, P and Q is allowed to vary by ± 1 from the current model;
- where p and q both vary by ± 1 from the current model;
- where P and Q both vary by ± 1 from the current model;
- where the constant c is included if the current model has $c = 0$ or excluded if the current model has $c \neq 0$.

Whenever a model with lower AIC is found, it becomes the new „current“ model and the procedure is repeated. This process ends when a model close to the current model with lower AIC cannot be found. The values of p, q, P, Q are not allowed to exceed specified upper bounds.

6.1.2 ETS Model selection

ETS model selection and parameter optimization works in a similar way. The selection process of ets() method implemented in R forecast package provides a general framework for model selection in an objective manner. The process begins with considering one ETS model at a time: ETS(X,X,X) where $X \in A, A_d, M, M_d, N$. Then the selection algorithm keeps altering the smoothing parameters of the model depending on previously detected time series features (e.g. seasonality period, variance etc.) and other logical rules [61]. The aim is to maximize either AIC, AIC_c or BIC metrics that give the probability that a specified outcome is generated defined as:

- **Akaike's Information Criterion (AIC)**

  It is based upon information theory, but a heuristic way to think about it is as a criterion that seeks a model that has a good fit to the actual time series values but few parameters [15].

  $$AIC = -2\log(L) + 2k,$$

  where L is the maximized value of the likelihood function for the estimated model, and k is the total number of parameters and initial states that have been estimated. L is calculated by fitting the model to the time series, it represents the likelihood that the data was generated by the model that we are investigating.

- **Bias Corrected Akaike's Information Criterion (AICc)**
  The AIC corrected for small sample bias (AICc) is defined as

  $$AIC_c = AIC + \frac{2(k+1)(k+2)}{n-k-2},$$

  where k is the number of parameters in the model, and n is the sample size. As n gets larger, AIC_c converges to AIC, and so there's really no harm in always using AICc regardless of sample size.

- **Bayesian Information Criterion (BIC)**

  $$BIC = -2log(L_k) + klog(n)$$

  Where L is the maximum likelihood estimation, k is the number of parameters in the model, and n is the sample size. As with the AIC, minimizing the BIC is intended to give the best model. The model chosen by BIC is either the same as that chosen by AIC, or one with fewer terms.

The algorithm implemented in R forecast package optimizes AIC_c by default. In the case that the data set contains zero values, then the multiplicative models can be omitted, which prunes the search space and saves a lot of computation.

6.1.3 Generalized Model selection

The general Box-Jenkins methodology - or Box-Jenkins Process - includes four stages as desribed in [13] and shown on figure 6.1:

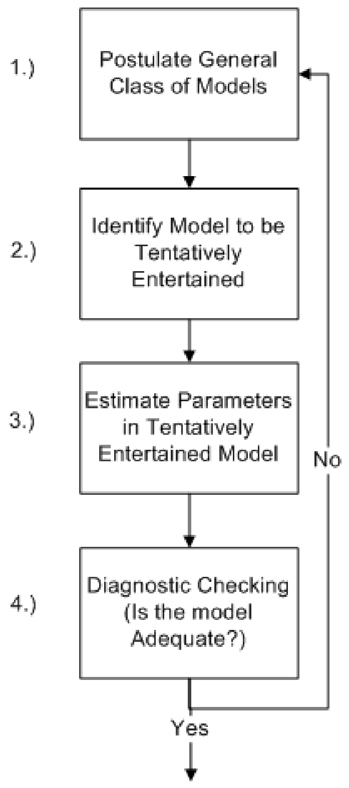

1.)
Postulate General Class of Models
2.)
Identify Model to be Tentatively Entertained
3.)
Estimate Parameters in Tentatively Entertained Model
4.)
Diagnostic Checking (Is the model Adequate?)
No
Yes

Figure 6.1: Original Box-Jenkins methodology [13]

1. Based on the time series we decide on a general class of models.
2. Identification of a model that can be tentatively entertained.
3. The order of the model and the level of difference are selected. Based on the time series, the model parameters are estimated.
4. The model is investigated with diagnostic checking tools. If it is not adequate the process is repeated with different model configuration. Otherwise if the model is adequate, it can be used for forecasting.

6.2 Classification Overview

The suggested classification process is shown in 6.2. The core of the process is the box-jenkins methodology, although there are some extensions. The crucial point is to consider the input time series as continuous stream. The process has to provide flexibility and fast reaction, in case the deviation between forecasted and measured values becomes high. Moreover, speed is an important point. The faster the system gives the forecast values the faster it can optimize the scheduling. There are possible dimensionality reduction techniques to speed up the classification and possibly the forecast process.

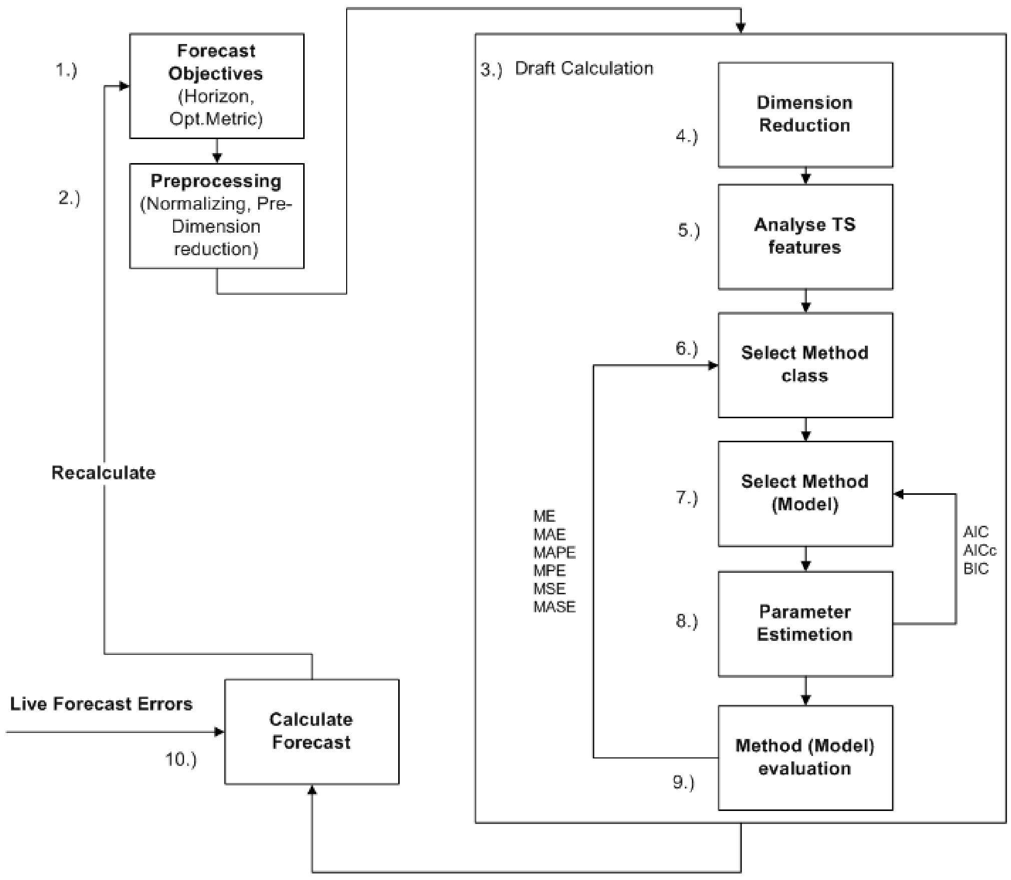


Figure 6.2: Proposed Classification Process

1. The process starts with the definition of the forecast objectives, these include choosing the intended optimization horizon, choosing the accuracy metric as the base of method comparison, and choosing the length of the input. The optimization horizon is not necessary the same as the forecast horizon, it is merely the interval used to calculate the chosen accuracy metric. This is one way to compare short term and long term prediction models. A choice has to be made between the accuracy metrics listed in section 5.8, based on which the comparison of different methods will be made. The choice depends on the

type of monitored entity, for example in the case of CPU usage it makes sense to choose a relative accuracy metric like MAPE or MPE that is based on proportional errors between forecasted and observed data. Whereas in memory usage traces, an absolute accuracy metric is proposed (like MAE, RMSE) because of the fact that usually applications need a fixed number of memory megabytes and changes in proportion of the overall amount of memory is less important.

2. Preprocessing: These are the steps to prepare the input time series for the forecasting procedure. This includes the elimination of missing values, the adjusting of shifted sampling periods, data manipulation and prior dimensionality reduction, (see section 6.4) so that the system has fewer data points to work with.

3. Draft calculation: To ease the classification process additional dimensionality reductions may be applied so that the comparison of the forecast accuracy, using different methods and models, goes faster.

4. Dimensionality Reduction: the first stage in the draft calculation is to apply dimensionality reduction (see possible methods in section 6.4).

5. Analyze Time Series Features: examining time series features like seasonality, variance, ACF plots, etc, will narrow down the possible search method classes. See figure 6.3.

6. Select Method Class: In this stage the choice is made whether the system applies seasonal long term forecasting or trend/level prediction. Refer to Figure 6.3 here to see the possible choices between method classes.

7. Select Method/Model: Iteratively, a method is selected and the desired metric accuracy is calculated.

8. Parameter estimation: some models like ARIMA and ETS have additional parameters to optimize. In this stage iterative parameter optimization is the goal.

9. Method (Model) evaluation: this stage serves as the comparison of the calculations and the forecasting methods. This stage also applies diagnostic tools to evaluate if the chosen method/model is indeed adequate.

10. Calculating the forecast: the selected approach will be applied. The resource usage is still monitored so that the system has the opportunity to compare the actual resource usage with the predictions. In the case of high error rates, this step adjusts the calculations, dimensionality reduction or initializes complete reclassification.

Figure 6.3 shows the suggested decision logic that classifies traces and assigns the most appropriate method class, method and/or model for forecast calculation. Earlier, an explanation of how to check seasonality and fluctuation of time series was touched upon.

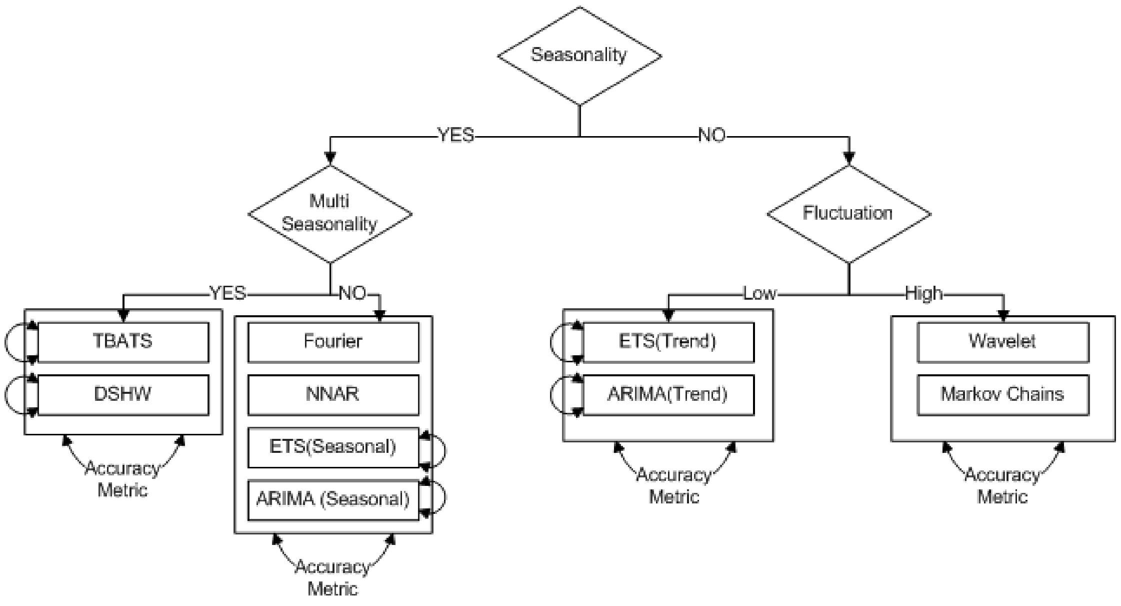


Figure 6.3: Classes of traces based on forecastability

6.3 Diagnostic Tools

Seasonality. Seasonality behavior and seasonal periods can be estimated using ACF, PACF functions (see section 2.3.1). In figure 6.4 ACF PACF was applied on the generated 3 days of cpu resource usage test data. Exponential decay in the seasonal lags on the ACF plot and a single significant spike at 1440 (1 day) in PACF can be seen.

Fluctuation. Here the signal burstiness with low-pass filtering explained in section 2.3.2 can be analyzed. Alternatively, it can be analyzed based on the decision logic in calculating short term (e.g. 1 hour) variance.

Forecast Diagnostics. The question that needs answering is whether the chosen method and model is adequate for forecasting or would it be better off without the forecasting and instead using a reactional scheduling procedure. To answer this questions the forecast results must be evaluted. Common evaluation procedures are based on the analysis of forecast residuals: residuals are the difference between the actual values and the predicted values, written as [20] [61]:

$$e_i = x_i - \hat{x}_i$$

If the model is correctly specified and the parameter estimates are reasonably close to the true values, then the residuals should have nearly the properties of white noise. They should behave roughly like independent, identically distributed normal variables with zero means and common

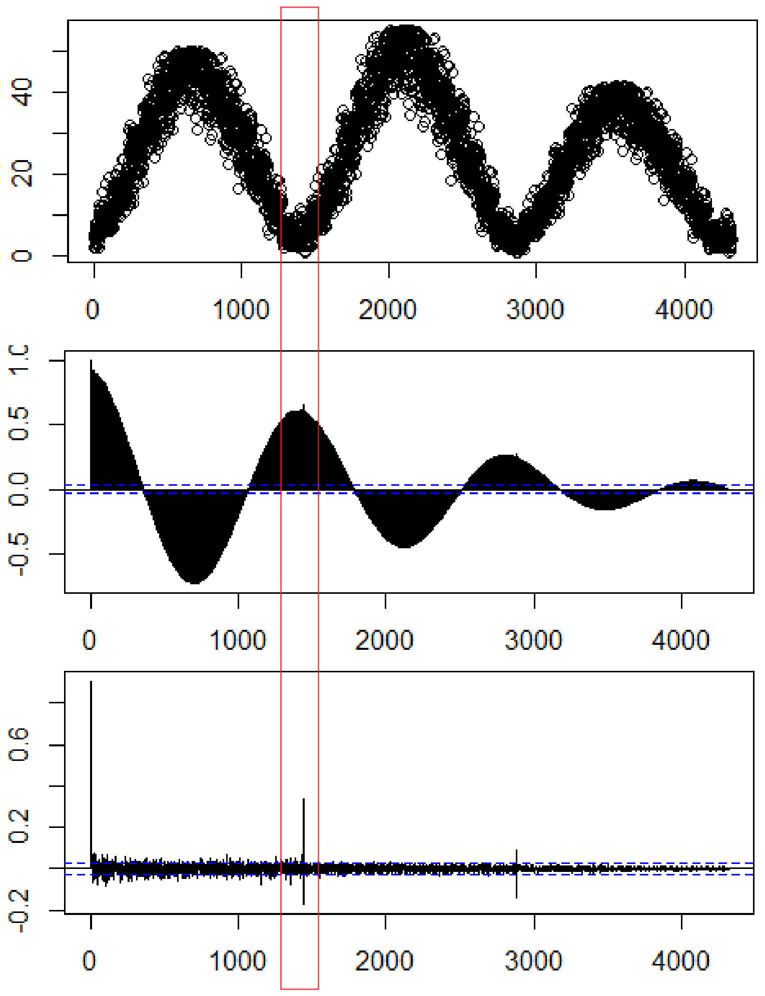


Figure 6.4: ACF PACF Behavior on the 3 day CPU usage trace test data

standard deviations. Deviations from these properties can help us discover a more appropriate model [20]. Residual correlation means that the forecast is biased. Theoretically, any forecasting method that does not satisfy these properties can be improved, since in this case, there are some inter-value correlations that are not used in the forecast model. That does not mean that forecasting methods that satisfy these properties cannot be improved. It is possible to have several forecasting methods for the same data set, all of which satisfy these properties. Checking these properties is important to see if a method is using all of the available information well, but it is not a good way for selecting a forecasting method. In addition to these essential properties, it is useful (but not necessary) for the residuals to also have the following two properties.

- The residuals have constant variance.
- The residuals are normally distributed.

However, a forecasting method that does not satisfy these properties cannot necessarily be improved [61].

One way to check for these properties is to analyze the plot, and the histogram of the residuals, or the ACF function of the residuals, to evaluate cross correlation.

In addition to looking at the ACF plot, a more formal test for autocorrelation can be done by considering a whole set of r_k values as a group, rather than treating each one separately.

r_k is the autocorrelation for lag k. When looking at the ACF plot to see if each spike is within the required limits, we are implicitly carrying out multiple hypothesis tests, each one with a small probability of giving a false positive. When enough of these tests are done, it is likely that at least one will give a false positive, so we may conclude that the residuals have some remaining autocorrelation, when in fact they do not.

In order to overcome this problem, it must be tested whether the first h autocorrelations are significantly different from what would be expected from a white noise process. A test for a group of autocorrelations is called a *portmanteau* test, from the French word describing a suitcase containing a number of items.

One such test is the **Box-Pierce** test based on the following statistic

$$Q = T\sum_{k=1}^{h} r_k^2,$$

where h is the maximum lag being considered, T is number of observations and r_k is the autocorrelation for lag k. If each r_k is close to zero, then Q will be small. If some r_k values are large (positive or negative), then Q will be large.

A related (and more accurate) test is the **Ljung-Box** test based on

$$Q^* = T(T+2)\sum_{k=1}^{h}(T-k)^{-1}r_k^2,$$

where h is the maximum lag being considered, T is number of observations and r_k is the autocorrelation for lag k. Large Q^* values suggest that the autocorrelations do not result by white noise series.

6.4 Dimensionality Reduction

Piecewise Aggregate Approximation (PAA) The time series data $X = x_1..x_n$ can be seen as one point in an n dimensional space. PAA is one of the simplest ways to reduce dimensionality: it divides the data space into N equal sized windows and in each window the mean value is calculated, and a vector of these values becomes the dimensionality reduced representation of the data. The process is visualized below [42].

Symbolic Aggregate approximation (SAX) SAX is an extension of PAA: first the averaged value for each window in calculated and within certain boundaries the value will be mapped to show specific symbols like the Figure shows below. Data is first normalized, and two steps

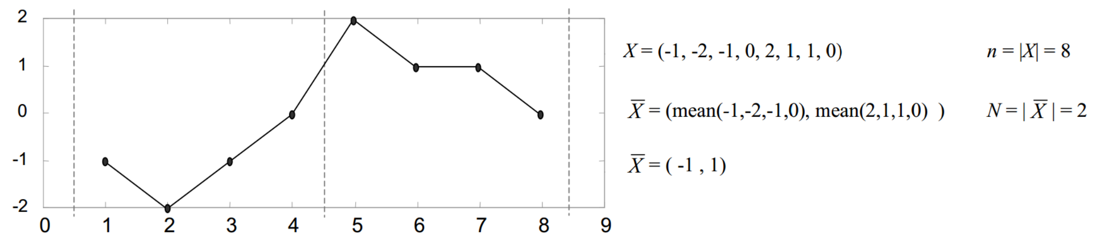


Figure 6.5: Illustration of PAA: original dimension n=8 is reduced to, N = 2 by averaging data in the two windows [42].

of discretization are performed. First, a time series T of length n is divided into w equal-sized segments; the values in each segment are then approximated and replaced by a single coefficient, which is their average. SAX is data adaptive since the boundaries of each symbol can be customized for the data set to which they are applied [49].

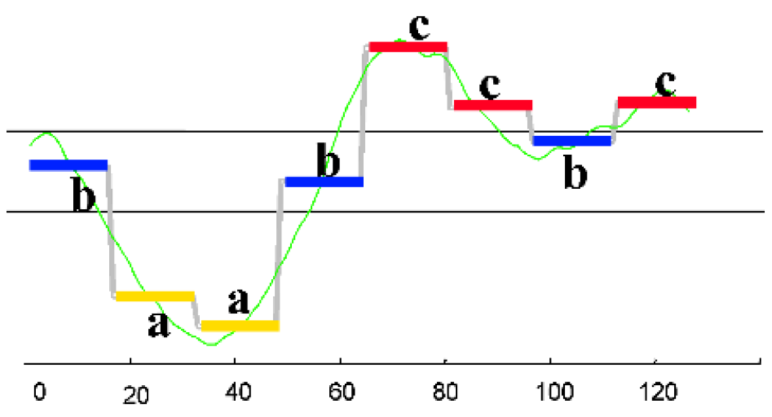


Figure 6.6: Example SAX representation. Time series of dimensionality n = 128 is mapped to the word = baabccbc (N = 8) [49]

Piecewise Linear Approximation (PLA) Piecewise Linear Approximation is a method used for approximating the original signal with piecewise linear segments as shown in the figure below. The crucial question is to choose the optimal number of segments used for the time series representation. This problem is clearly a trade-off between accuracy and dimension reduction [49].

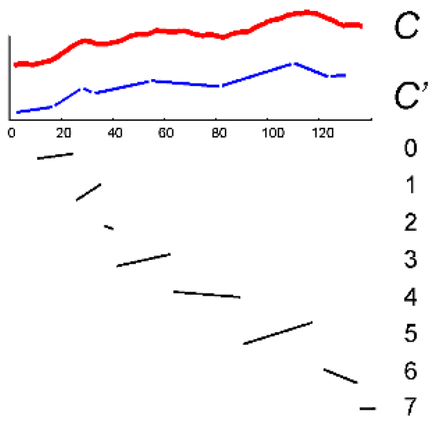


Figure 6.7: Visualization of PLA [49]. C represents the original time series. The original time series is segmented and each segment is represented by a linear function (0-7), this results in the approximation C'. After reducing the time series sample rate, PLA can be used to interpolate between the sample points and reduce the data loss caused by dimensionality reduction.

CHAPTER 7

Cloud Simulation

In this chapter the prototype of our cloud simulator is introduced. This uses forecasting to prevent SLA violations in a proactive way. First, we describe the resource allocation problem and the included challenges. We also describe in detail, how existing reactional cloud scaling systems are trying to give a solution. Then the way in which forecasting can enhance these reactional scaling systems is discussed. Finally, the simulation results are presented in this chapter.

7.1 Introduction

The next stage of the thesis is to evaluate forecasting methods applied on the monitored resource traces. In particular, how efficiently they can be used to optimize the resource allocation process in large scale distributed systems [72]. Resource allocation in cloud computing is the process of assigning available resources to the cloud applications. Resource allocation strategy is about integrating cloud provider activities for utilizing and allocating scarce resources within the limit of a cloud environment so as to meet the needs of the cloud applications. An optimal Resource allocation strategy would avoid:

- Over-provisioning: arises when the application gets surplus resources then the demanded one.
- Under-provisioning: occurs when the application is assigned with a fewer number of resources then the demand.

In other words, an optimal resource allocation strategy would allocate exactly enough resources to run the cloud applications as needed to keep „fair“ Quality of Service (QoS). The question is how much resources are needed to guarantee fair QoS, and what does „fair“ QoS mean. In most cloud computing systems today, QoS parameters such as throughput, latency and response time, are specified in a standardized form of agreement between cloud provider and cloud users.

The idea of fairness among cloud applications is to never co-locate applications so that their resource needs exceed the physical boundaries of the machine on which they are deployed. As discussed earlier, QoS metrics such us response time, remain under a reasonable limit as long CPU usage has not reached 100% (see Figure 4.6).

Cloud resources consist of physical and virtual resources. Generally, today's multicore processor systems allow the isolated hosting of multiple Virtual Machines (VM) on the same Physical Machine (PM). Additionally, on each VM one or more applications can be deployed. The VM resources can be scaled up and down as long the physical boundaries keep up. Additionally, VMs can be started, turned down, be migrated, or cloned during the allocation process. The main goal is to optimize Resource Allocation so that the VMs are dynamically rescaled based on the application's resource needs, without exceeding physical hardware limits. If the physical limit is exceeded that means that on at least one VM, there is an application running that needs more resources than it has been provided. This causes QoS degradation. Ideally, forecasts should be used to eliminate as much of these violations as possible.

Giving applications exactly „as much they need" without under-/over- provisioning, has the main advantage of maximizing the overall physical utilization in the cloud. This maximizes the profit for providers while guaranteeing consumer satisfaction. Maximizing utilization also means that unused physical machines can be turned off. This saves electricity and cuts amortization costs too.

A cloud simulator was implemented to compare the usage of seasonal forecasting, to prevent physical limit violations. This chapter is dedicated to explaining the simulation results.

7.2 Dynamic Resource Scaling

7.2.1 Dynamic server pool scaling

When it comes to increasing or decreasing application resource demand, the elasticity in cloud systems is provided by the following operations [54]:

- Scaling VM Resources: Several cloud architecture allows one to add or subtract resources for running VMs „on-the-fly". For example, if one application running on a VM needs more RAM in order to keep up with the demand, the system can just provide it, if the physical hardware is available.
- Start new VM from scratch: create a new VM and start the Operating System and application from the beginning.
- Cold cloning: create a snapshot of the application VM beforehand and then instantiate a new server using the snapshot.
- Hot (Live) cloning: entails cloning the source machine while it is running it's operating system.
- Additional tools exists to ease the process of VM cloning by flexible state transferring: various VM states will be archived using VM descriptor containing metadata, and transferred on-demand. See project SnowFlock for further details [14] [56].

7.2.2 Reactional scheduling

Elastic capabilities offered by cloud providers are typically cast as rule-based algorithms that define scaling conditions based on a target metric reaching some threshold [52]. For example, cloud providers such as Amazon [3] or third party tools such as RightScale [60] or AzureWatch, [7] offer rule-based methods for auto-scaling.

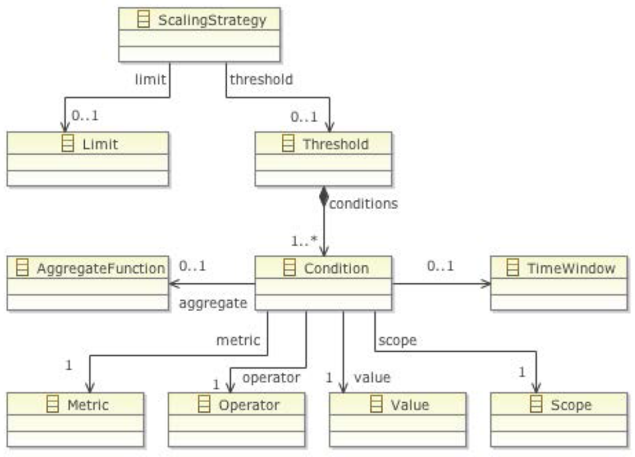


Figure 7.1: Scaling rule (R) meta-model [52]

Figure 7.1 shows a meta-model schematic for the scaling rule R. Rule R is used by the system to trigger scale in or out, R consist of the *Scope*, *AggregateFunction*, (e.g. average, median), a metric(Q), a defined *TimeWindow* (T_1), an *Operator* (e.g $>, <, =$) and some *threshold value*. The administrator can choose for example to aggregate the CPU usage in an arbitrary scope. Then, if the aggregation over the past e.g. 10 minutes is more then the threshold value, the scaling process is triggered. Rule R also specifies a 'Limit' on the minimum (for scale in) or maximum (for scale out) number of instances within the tier.

To be more concrete, Amazon, for example, lets the users define specific rules for auto-scaling: if latency on one cloud instance exceeds 4 seconds in any 15 minute period, *Amazon AutoScale* automatically adds a new instance behind the *Elastic Load Balancer* [3].

One of the biggest cloud providers works based on these kind of elastic auto scaling rules. The problem with this reactive approach is that the reaction to the changing resource demand is slow. As seen in the example, detecting it takes 15 minutes, starting a new cloud instance takes a couple of more minutes, and in this time interval, QoS degradation is almost always guaranteed. This thesis proposes the usage of resource demand forecast values to derive additional rules in such systems, for initializing proactive resource scaling. Proactive resource allocation leads to

- Fair Quality of Service,
- While maximizing utilization,

- Thus cutting electricity cost by turning down unused resources, or maximizing the number of deployed cloud applications
- Without exposing too much additional computation to the cloud. The most expensive part of the process is monitoring the instances, which is already done by reactive systems.

7.3 The Scheduling Problem

The goal is to enhance scheduling in cloud computing systems beyond the limits of reactional auto scaling solutions. It is desirable to achieve better system utilization in order to use minimal amounts of machines to host all cloud applications. Figure 7.2 visualizes the problem: there are n Applications waiting to be hosted in the cloud. Each application needs some resources in order to be hosted, e.g., CPU, RAM. Each application has to be hosted on a physical machine, although they have limited resources. Cloud scheduler has the duty to take an application and assign it to one of the machines that has enough resources to host it.

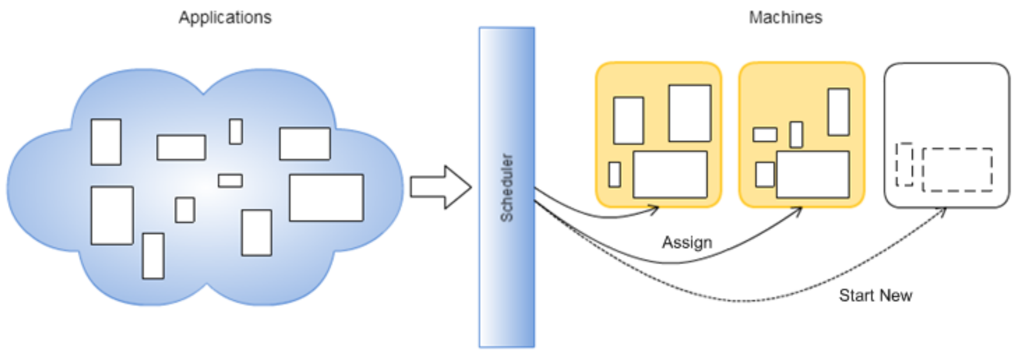


Figure 7.2: Scheduling visualized as 2D bin packing

Figure 7.2 shows the problem as a 2D bin packing problem. In this context the x-axis represents the CPU need/capacity and the y-axis represents the RAM need/capacity. Scheduler has to pack the fixed size bins (machines) so that their limit is not exceeded and the goal is to use as few bins (machines) as possible. Calculating the optimal assignment is known to be an NP-hard problem. If the resource usage of applications would be static, calculating the optimal assignment would still be feasible. Although, additionally the resource needs of each application changes in every second. In order to find the optimal assignment, one would need to calculate a NP-hard bin packing problem every second, not to mention that if an optimal assignment is found, the migration of the applications is also not free. This problem can be extended by adding more resource need/capacity dimensions: so far two dimensions have been used in the example (CPU, RAM), but it can be extended to include several other dimensions. In the simulation 4 dimensions are used: CPU, RAM, bandwidth (alias NET), and disk I/O. Becuase of the complexity of the problem, one can hope to come up with sophisticated heuristics at best. In order to make sure that an applications QoS is not violated, a naive scheduler would assign applications based on their historical maximum resource usage, which results overprovision. This is why violation detection in cloud computing systems is very important and an accurate SLA detection method would increase utilization in the cloud.

7.4 Using The Forecast Values

There are several different approaches to using the forecasted values to optimize the cloud computing scheduling process. This chapter describes some of those approaches.

7.4.1 Using Forecast to prevent SLA violations

This is one of the first things that comes to mind when enhancing a reactional system: In addition to existing reaction rules, the system is extended with rules that are based on the predictions. The result is a hybrid resource allocation system that works based on predictive and reactive rules. The advantage of such system is that - depending on the forecast accuracy - a high percentage of the SLA Violations can be predicted in advance, so the scaling process (turning on additional machines, providing more resources to existing VM, etc..,) can be initialized earlier, before entering into the phase of QoS degradation. Such hybrid resource provision systems are described in [54] [52]. This rule based prevention is simulated in the implemented cloud simulator (see 7.5).

7.4.2 Using Forecast to find optimal migration target

In addition to SLA violation prevention rules, having forecasts of each application's future resource needs available, allows one to optimize the selection of the machine which one wants the application to migrate to. Migration is an expensive process, therefore one should not migrate an application to another machine from where it should be also soon be migrated. Having knowledge about the future resource needs of an application and the other cloud nodes (machines), allows for optimization of the selection rules of the migration target.

Such selection rule is to use the machine with the maximum euclidean resource trace distance:

In Figure 7.3 A denotes the forecasted time series of an application, M denotes the forecasted time series of a machine (sum of applications forecasts). The distance between the two series is given by the euclidean distance between each value:

$$d(A, M) = \sqrt{\sum_{i=1}^{n} (a_i - m_i)^2}.$$

This distance is low when the trace periodicity is similar and the time series are moving together, and high if the two signal complement one another. When looking for complementing

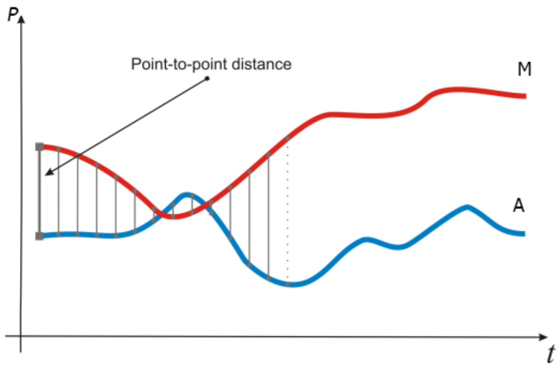


Figure 7.3: Euclidean distance between time series

traces, the recommended migration target is the machine with the maximum trace distance, see Figure 7.4.

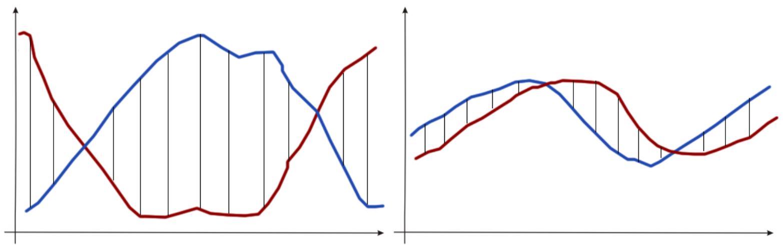


Figure 7.4: Euclidean distance is high when seasonal similarity is different.

Each time series is normalized to have a mean of zero and a standard deviation of one before calling the distance function, since it is well understood that in virtually all settings, it is meaningless to compare time series with different offsets and amplitudes [47].

The calculation can be optimized with dimensionality reduction: there is no need to calculate the distance for every point. The calculation can be reduced to every k-the data sample. This will result in some data loss, but still deliver a good decision heuristic.

Another approach would be to keep track of the resource usage trend of the physical machines in the most recent sliding window. Then, for migration target select a machine whose usage shows decreasing trend.

7.5 The Simulator

The implemented cloud simulator uses seasonal forecasting methods (described in chapter 5) to detect physical machine limit violations before they happen. The way to proceed is to:

1. input historical traces of applications (e.g. 3 days)

2. train-test split : split the input e.g.: calculate forecast based on 2 days, the third day is the test.
3. start machines with fixed capacity, assign applications
4. iterate time (traces are changing)
5. every time there is a violation in a resource limit, we check whether is was predicted by the calculated forecast.
6. scale the cloud in, in order to make more violations happen (consolidate applications if possible).

7.5.1 The Input

The workload generation is described in depth in the chapter 4. While observing resource usage changes of several applications over time, the daily periodicity can be seen clearly (See Appendix A). The generated workload traces intend to recreate these periodical fluctuations. The workload is generated synthetically in a controlled environment. While hosting an instance of the popular content management system drupal on a VM, it was requested by emulated clients. The client threads requesting the drupal workload were generated by JMeter. The number of threads, and so the number of requests were increased stepwise during the generation and then decreased, similarly as the number of users increasing and decreasing during a typical day. During the simulation various resource usages of the hosting VM were monitored. The generated workload contains one day of usage traces where the usage rate simply goes up during the day and then down again. This one day trace is then concatenated 3 times, for each day additional randomized scaling is applied and plotted on figure 7.5.

- The concatenated trace contains 3 day logs of 4 dimensions: CPU, Memory (MEM) , Network usage (NET), Disk I/O Operations (DISK).
- Each dimension contains all together 4320 data points. This corresponds to 1440 data points per day.
- Thus, each data point represents the amount of resource usage averaged for a 1 minute interval.

The generated test data has certain weaknesses. High variance is a result of the fact that only 10 clients could be emulated during the workload generation process before the cpu usage hit the top limit and each simulated client sent their request randomized. Additionally, to ease the test data generation, the actual traces were obtained by per second monitoring intervals observed to be 24 minutes long. Another weakness is that the seasonality behavior is very simple and this gives an advantage to simple forecasting methods. In the memory traces the sudden drop after each day is caused by the concatenation: at the end of the workload generation process every resource usage value goes back to the same state as before, but the memory is still occupied since there is no need to empty it.

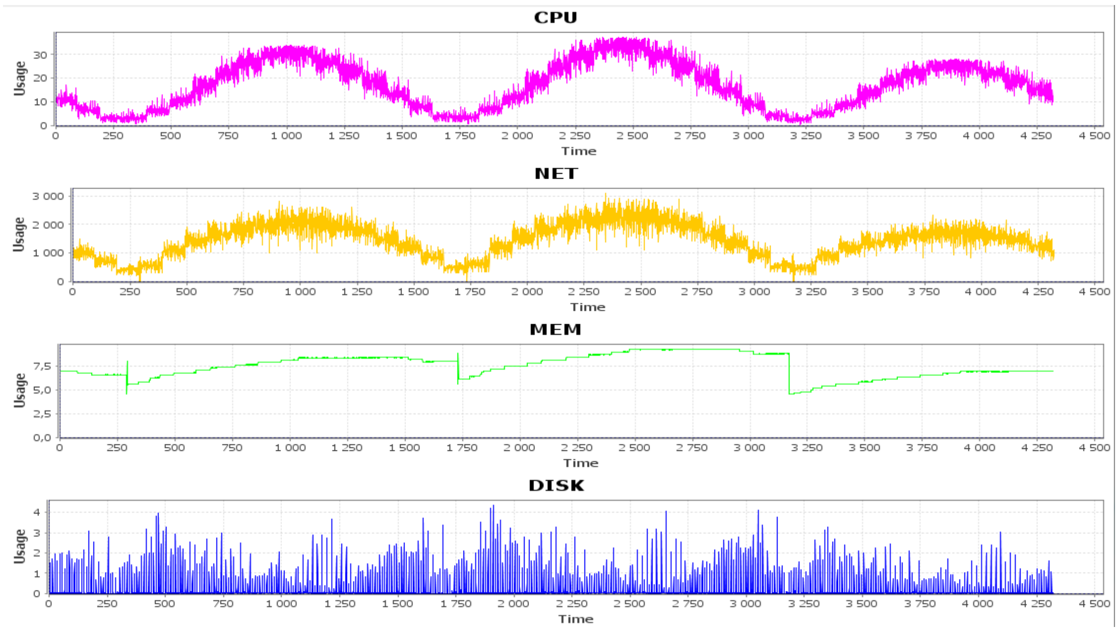


Figure 7.5: CPU, MEM, DISK, NET traces visualized

Knowing these weaknesses the input of the simulation is kept flexible. The simulator can work with any input feed from CSV files.

To simulate the whole cloud, more than one application trace is needed. Therefore, over 200 traces based on the following assumptions are generated:

- In a large scaled cloud there are (web) applications running used by people in different countries and in different time zones. Therefore our assumption is that cloud applications have differently shifted periodic/seasonal behaviors during the day, e.g. a web application targeted to german speaking regions may have the peak up to 8 hours earlier than the web applications targeted for US citizens because of the time zone difference. Another cause of seasonality shifting could be caused by different working times in various companies (in case of cloud hosted business applications).

- Applications have different proportion of resource usage needs. E.g some of them needs more cpu, others more diskIO.

The trace generation for cloud applications is visualized in Figure 7.6.

- The traces are shifted horizontally in order to simulate the different seasonality.

- The traces are scaled vertically in order to simulate different resource usage proportions.

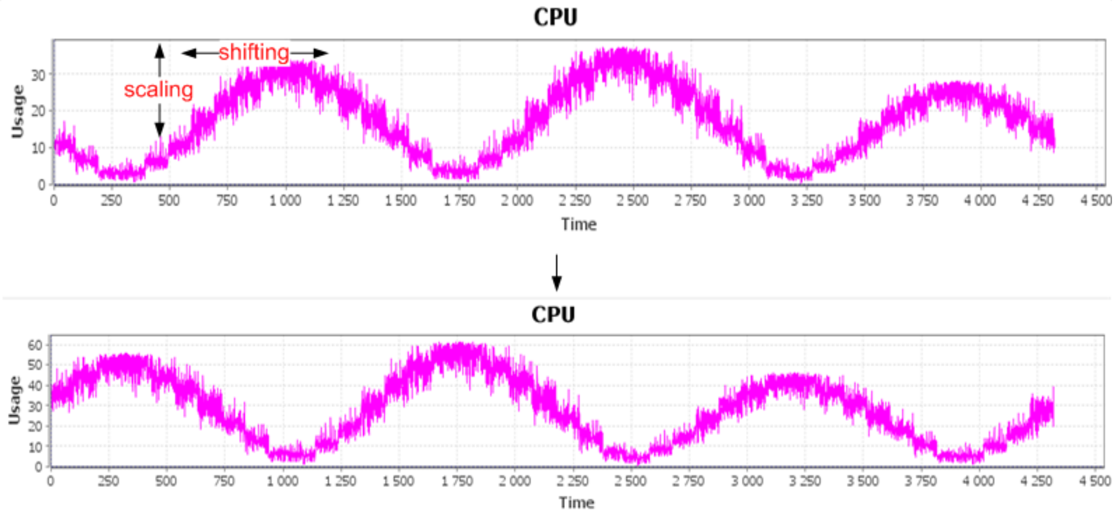


Figure 7.6: Producing the workloads for the cloud

7.5.2 Implementation Tools

Java

The cloud simulator is implemented in the high level object-oriented programming language Java. Java is a popular language because of it's platform independence and the fact that it is open source and free. The main reason to use java for implementation is because the open source community offers a lot additional well documented tools to extend basic language functionality. These tools allow one to create helpful visualizations about the used data and the forecasts, and easy data manipulation. Additionally, using java enables one to create easy to understand code that can be extended and customized in the future [57].

JFreeChart

JFreeChart is a free Java chart library that makes it easy for developers to display professional quality charts in their applications. It is open source and well documented. JFreeChart was used to provide meaningful visualizations about the input data such us the resource usages of each separate application. It is also used to provide a visual comparison of the used forecasting tools [46].

R

R is an integrated suite of software facilities for data manipulation, calculation and graphical display. R is a free programming language, used widely for statistical computing, data mining, and data analysis. The development community behind R offers a large collection of packages, each package includes several functions and algorithms to be used. R is an interpreted language; users typically access it through a command-line interpreter [63].

R Forecast package

Developed by Rob J Hyndman, professor of statistics at Monash University in Australia. Rob J Hyndmann is a domain expert in the field of statistical forecasting. Under his development the R forecast package aims to extend basic R features, providing methods and tools for displaying and analyzing uni-variate time series forecasts, including exponential smoothing via state space models and automatic ARIMA modeling. The simulator implemented uses many of the algorithms implemented in this package [62].

JRI

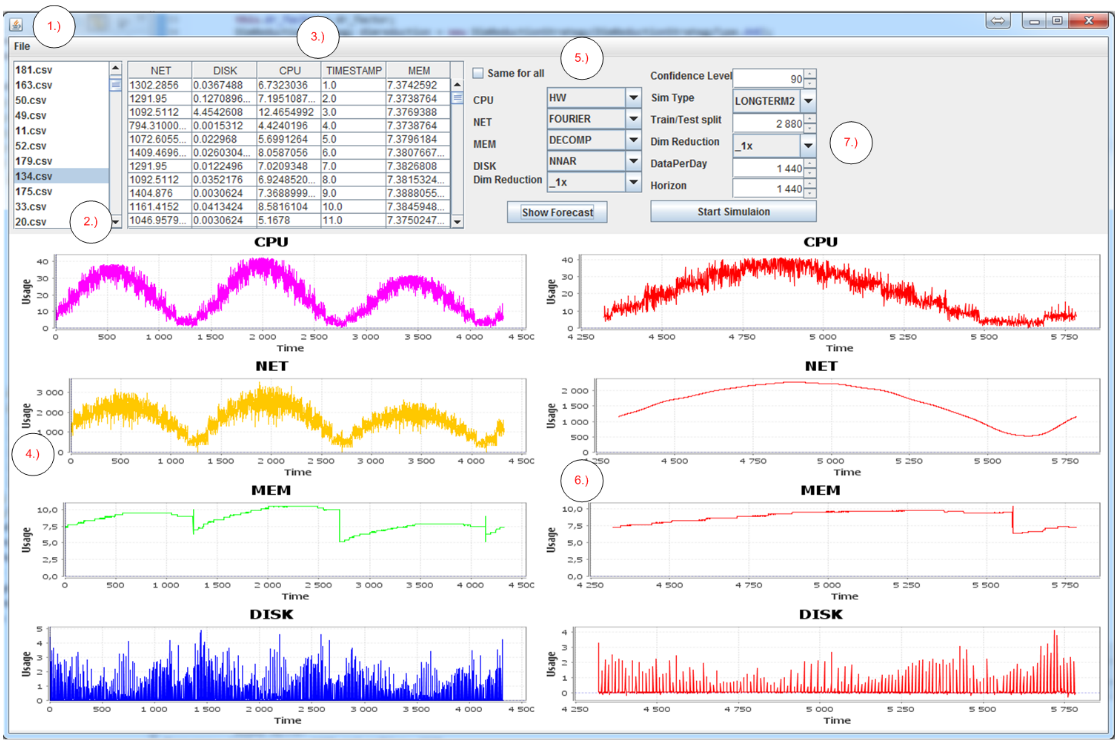
In order to use the R forecast package and other R functionalities from the java environment an integration tool is needed. JRI is a Java/R Interface, which allows R to be run inside Java applications. Basically it loads R dynamic library into Java and provides a Java API to R functionality. It supports both simple calls to R functions and easy access to R data structures. In order to complete the integration *rJava* package has to be installed in R, which allows one to create objects, call methods and access fields of Java objects from R [71].

7.5.3 Implementation

Figure 7.7 depicts a screenshot of the simulator.

It consists of the following parts:

1. Input Selection: the input directory, containing the CSVs of the applications traces, needs to be selected.
2. Application List: when the loading of application traces is finished, their name is listed in this panel.
3. Data Browser: the data browser serves for inspecting the data rows of the selected application traces.
4. Visualization of the input: the selected application traces are dynamically plotted when selected.
5. Forecast panel: for each dimension an individual forecast method can be selected, with optional dimensionality reduction. Before running the simulation the forecasts can be plotted and examined.
6. Visualization of Forecasts: This area shows the results of the forecast methods selected on panel 5.) applied on the selected application trace. In this example different methods on each dimension are used: Holt Winters exponential smoothing on CPU, Fourier forecasting on NET, STL decomposition based forecasting on MEM, and Neural Network Autoregression on DISK.



181.csv
163.csv
50.csv
49.csv
11.csv
52.csv
179.csv
134.csv
175.csv
33.csv
20.csv

NET	DISK	CPU	TIMESTAMP	MEM
1302.2856	0.0367488	6.7323036	1.0	7.3742592
1291.95	0.1270896...	7.1951087...	2.0	7.3738764
1092.5112	4.4542608	12.4654992	3.0	7.3769388
794.31000...	0.0015312	4.4240196	4.0	7.3738764
1072.6055...	0.022968	5.6991264	5.0	7.3796184
1409.4696...	0.0260304...	8.0587056	6.0	7.3807667...
1291.95	0.0122496	7.0209348	7.0	7.3826808
1092.5112	0.0352176	6.9248520...	8.0	7.3815324...
1404.876	0.0030624	7.3688999...	9.0	7.3888055...
1161.4152	0.0413424	8.5816104	10.0	7.3845948...
1046.9579...	0.0030624	5.1678	11.0	7.3750247...

Same for all

CPU	HW
NET	FOURIER
MEM	DECOMP
DISK	NNAR
Dim Reduction	_1x

Confidence Level	90
Sim Type	LONGTERM2
Train/Test split	2 880
Dim Reduction	_1x
DataPerDay	1 440
Horizon	1 440

Figure 7.7: Simulator GUI

7. Simulation Panel: this panel serves for setting simulation parameters such us: **Confidence level**, **Test/Train split**: what percentage of the input data is used to train the forecast methods and which subset is used to simulate the actual application resource usage in the cloud. *Dimensionality Reduction*: to ease the calculations dimensionality reduction can be applied. **Data Per Day (or seasonal period)**: the number of data points that the input data contains per day, this gives the daily seasonal period. **Horizon**: how many data points one wants to forecast. In the example the input data consists of 3 days of resource usage logged every minute. Altogether that is 4320 data points. If one wants to train the forecast methods based on the first two days of usage data (2880 data points), it is necessary to define the forecast horizon as one day (1440 data points) and run the cloud simulation with the rest of the data (test set) (1440 data points).

The simulator components are shown in Figure 7.8 :

- Input Parser: The input is saved in Several CSV files each containing the resource needs of an application for the given time interval. Each row represents a state in a specific time interval (e.g. 1 minute). The parser converts the file into Time Series Model.

- Time Series Model: After the input is converted it is held by this internal data structure.

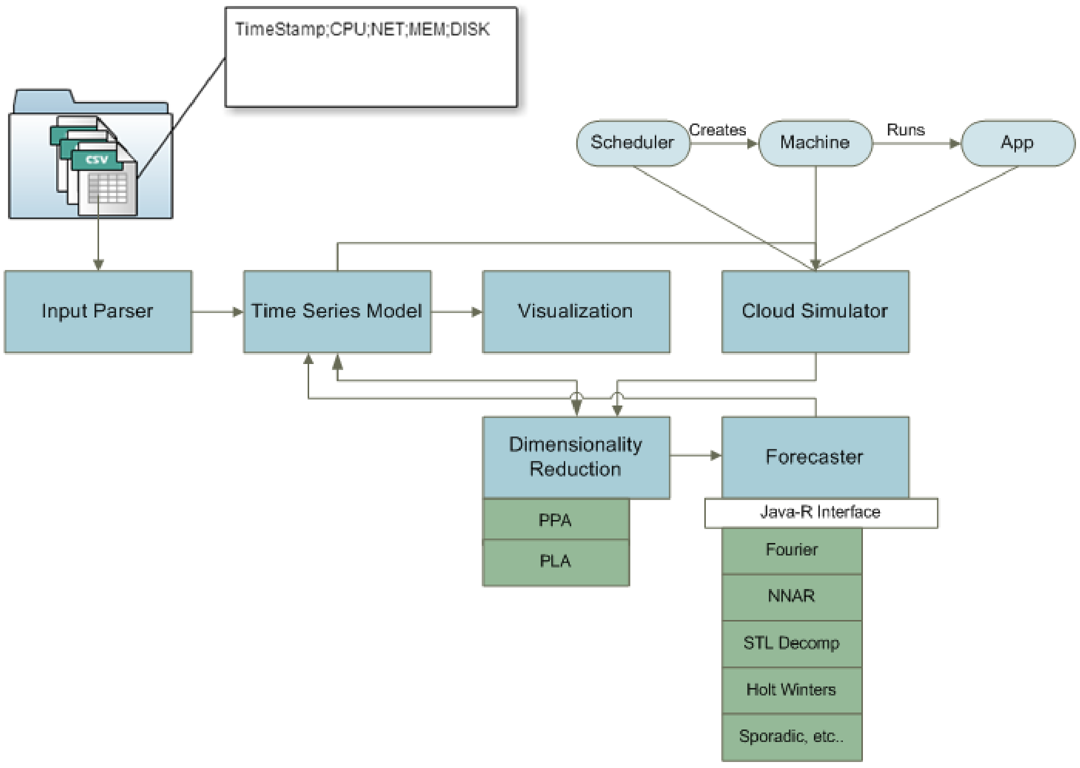


Figure 7.8: Simulator components

- Visualization : The simulator uses JFreechart to plot line graphs of usage traces and forecast.
- Dimensionality Reduction: Before visualization forecasting or simulation, certain dimensionality reduction processes can be applied: currently PPA, and PLA are implemented (see Section 6.4). The architecture of the simulator can be easily extended to other dimension reducing methods.
- Forecaster: This component serves to calculate forecast values. In the current stage the implemented methods communicate with the R forecast package [62] through java-R interface [71]. This part is also extendible for additional methods. The current implementation uses Fourier forecasting, NNAR, STL Decomposition based Forecasting, Holt Winters exponential smoothing, Sporadic forecasting with fourier, and some naive methods: MEAN, MAX, NAIVE seasonal.
- Cloud Simulator: this is the component that simulates the scheduling process in a cloud computing environment. It starts machines, assigns each application to a machine and reacts to changing resource needs.

7.5.4 Initial Workload Distribution

This section is dedicated to explain how the cloud scheduler simulator works. The input is converted to a set of applications that need to be assigned to one machine that has the capacity to host the application. Each machine has a fixed resource capacity (can be changed before the simulation starts). The scheduler makes an initial distribution of the application list, which starts as many machine as needed. Then the clock starts ticking, and as the time changes so do certain resource needs of the applications. The scheduler simulates a reactional scheduler: if there is a machine where the physical limit is violated, in other words the resource needs of the hosted applications exceed the physical limit of the machine, one or more applications have to be migrated from that machine to another machine. If there is no machine currently available in the cloud that could host those applications a new machine has to be started. This reactional scheduler is later extended to simply count the occurrences of limit violations and check if the forecast values of the selected forecasting algorithm are predicting the violation.

Figure 7.9 shows the process of the initial assignments of applications. Figure 7.10 shows the reactional scheduling process.

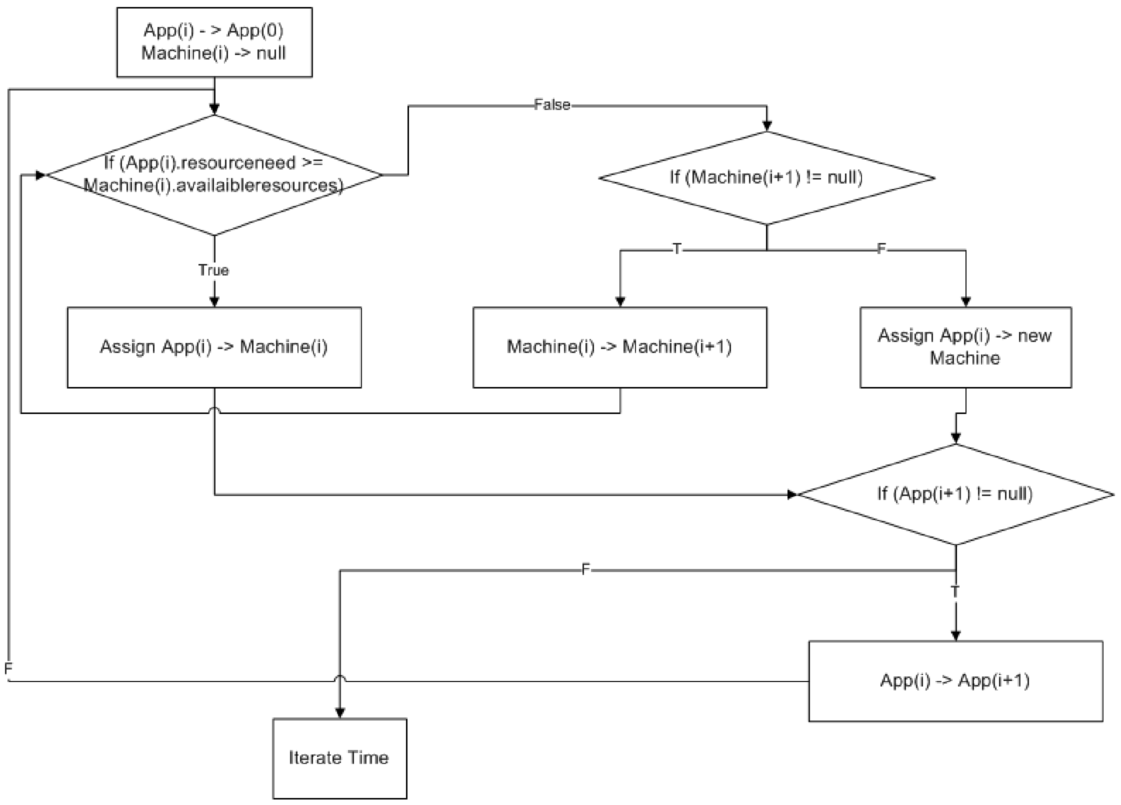

App(i) - > App(0)
Machine(i) -> null
False
If (App(i).resourceneed >= Machine(i).availaibleresources)
If (Machine(i+1) != null)
True
T
F
Assign App(i) -> Machine(i)
Machine(i) -> Machine(i+1)
Assign App(i) -> new Machine
If (App(i+1) != null)
F
T
F
App(i) -> App(i+1)
Iterate Time

Figure 7.9: Process of initial workload distribution

The process of the initial workload distribution is a first fit heuristic. The simulator iterates over the list of applications and assigns the current application to the first machine that has the available resources (for each resource need dimensions) to host the application. If such a machine is not found in the cloud, the scheduler simply starts a new machine, and assigns the current application there.

One can use many other heuristics to distribute the applications based on the prior knowledge we have about their resource usage such as the average/max CPU/RAM usage, etc. Such heuristics are further described in the paper [35].

Random assignment. Applications are assigned to any randomly selected machine that has the available resources to host the application.

Large to Small Applications. Applications are ordered based on some attributes such as average CPU usage, and assigned to cloud resources in that order.

Small to Large Applications. Applications are ordered and then assigned in reversed order: first applications with less resource needs are assigned, then the rest.

Assignment based on usage proportions. Knowledge of past resource needs can be used so that the proportions of the resource needs of an application and the proportions of the available resources of individual cloud nodes are inspected. For example, lets say an application needs 10% CPU and 70% Disk I/O. (Percentage is given by the maximum performance of one cloud node). One cloud node has available resources of 20% CPU, 90% Disk I/O, another cloud node has 90% CPU and 90% Disk I/O free. This heuristic would choose the first one to host the application, to maximize utilization.

Maximum Seasonal difference. Let's say there is a cloud node running an application with some seasonal behavior that peaks at 13pm. It make sense to co-locate an application on this cloud node that has it's minimum usage at that time. This way resource usage needs will more likely complement each other during the day. This heuristics needs prior knowledge about the periodic behaviors of the applications. The forecasted resource usage needs to be compared with the summed resource usages of the applications that are running on a specific cloud node. Then the euclidean distance between the two time series has to be calculated. If the distance is high, the usage needs will be more likely to complement each other during the next seasonal period. The steps are the follows:

1. Apply e.g. fourier forecast
2. Sum up the applications forecasted resource needs that are running on the same cloud node
3. Calculate the euclidean distance between the application forecast and the cloud node forecast.
4. Search for the cloud node that has the largest euclidean distance

5. Assign application to the found cloud node.

7.5.5 Migration simulation

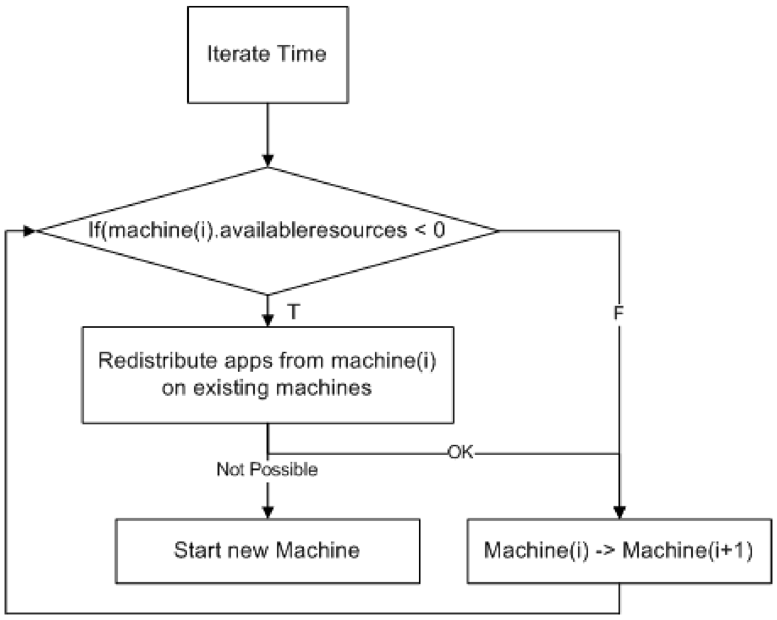

Iterate Time
If(machine(i).availableresources < 0
T
F
Redistribute apps from machine(i) on existing machines
OK
Not Possible
Start new Machine
Machine(i) -> Machine(i+1)

Figure 7.10: Process of simulation

Figure 7.10 shows the simulation of the migration process. Once the initial assignment of each application is finished, and all applications are assigned to at least one machine, the time is iterated. Application resource needs are continuously changing so physical limit violations can happen at any moment, and the simulator has to be ready to react to them. The reactional simulation simply calculates the resource usage at every given time for each machine. When it is more than the maximal machine limit, then as many applications as necessary will be relocated: they are either assigned to a different machine, that has enough resources to host them, or a new machine is started and the applications are assigned to that machine. During the process the number of started machines, the number of migrations (re-locations), and number of violations are counted.

The aim is to detect violations before they happen using statistical forecasting methods and simulation of proactive reaction rules. In order to maximize the number of violations in the system, after each iteration the simulation will try to shut down the computer with the minimal utilization: if each application that is assigned to a machine with minimal utilization can be assigned to a different machine, without directly causing violation, the simulator takes the opportunity, migrates the applications, and shuts down that machine. This way, cloud utilization is maximized during the simulation and so more violations are expected to take place, and be detected. See figure 7.11.

In this simulation a simple first fit heuristic is used.

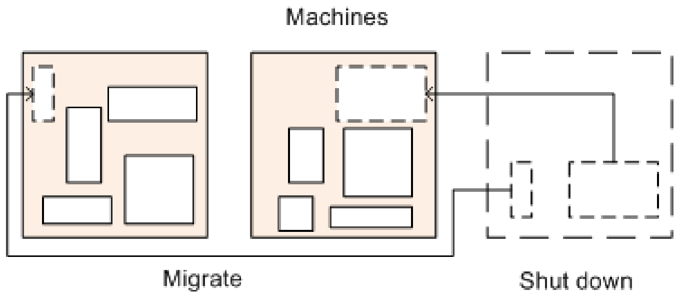


Figure 7.11: Shutting down the least occupied machine in the simulation

7.5.6 Simulation of violation detection rules

For the simulation, the reactional cloud simulator is extended with a violation detection: every time a violation happens (the needed resources are more than the available physical resources on some machine) the simulator will look on the forecasted resource usages and determine whether the violation was predicted by the forecast. The detection is explained in Figure 7.12.

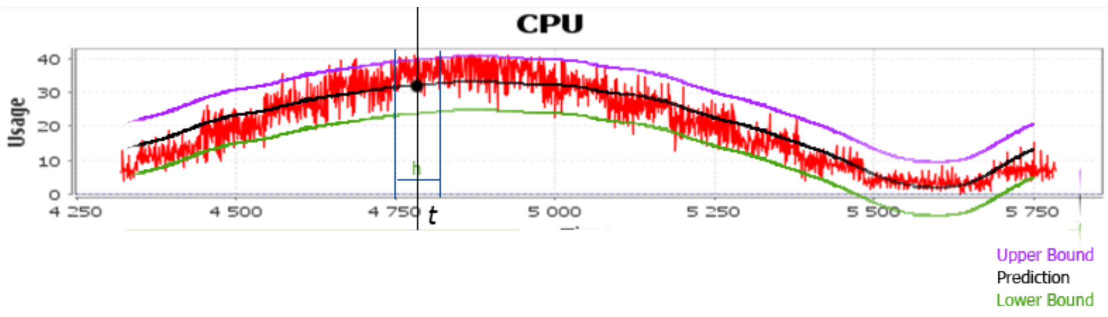


Figure 7.12: Using Forecast values to enhance scheduling

For each forecast point there exist a pessimistic (upper bound) and an optimistic (Lower Bound) prediction. Upper Bound is drawn with purple and lower bound with green line in the figure above. The optimistic and pessimistic estimation is controlled by the confidence level, it is calculated as follows:

$$U_i = F_i * (1 + ((100 - Confidence)/100)),$$

$$L_i = F_i * (1 - ((100 - Confidence)/100)),$$

U_i denotes the i-th element of the upper bound, L_i denotes the i-th element of the lower bound, F_i denotes the i-th element of the forecast values.

At a time t the simulator looks at the forecast values in a horizon h:

- If There is a violation **And** there is a pessimistic forecast that predicted it: consider it as **predicted**.
- If There is a violation **And** there is **no** pessimistic forecast that predicted it: consider it as **missed**.
- If There is **no** violation **But** there is an optimistic forecast that predicts a violation: consider it as **false positive**.

7.5.7 Results

- Machine Violation: Number of machines on some resource violation happened during the simulation.
- Machine Predicted: Number of machines on some violations happened, that is previously predicted by the forecast.
- Machine Missed: Number of machines on some violations happened, that is previously **not** predicted by the forecast.
- False Positive Machines: number of cases where machine violation is predicted by the forecasts but not happened.
- Dimension Violation: Summed number of dimensions that are violated on each machine during the simulation.
- Dimension Predicted: Number of dimension violation that are predicted previously.
- Dimension Missed: Number of dimension violations that are not predicted by the forecasts.
- False Positive Dimensions: Number of cases where dimension violation is foreseen by forecasts but not happened.
- CPU/MEM/NET/DISK Violation: Number of specific Dimension that is violated.
- CPU/MEM/NET/DISK Predicted: Number of cases when a specific dimension is violated and it is previously forecasted.
- CPU/MEM/NET/DISK Missed: Number of cases when a specific dimension is violated but not forecasted.
- CPU/MEM/NET/DISK False Positive: Number of cases when violation is forecasted but not happened.

The simulation results were obtained using 5 different forecast methods: Fourier forecast, STL decomposition based forecast, Neural Network autoregression, Holt winters exponential smoothing, and Double seasonal Holt Winters. Note that these methods are used for forecasting

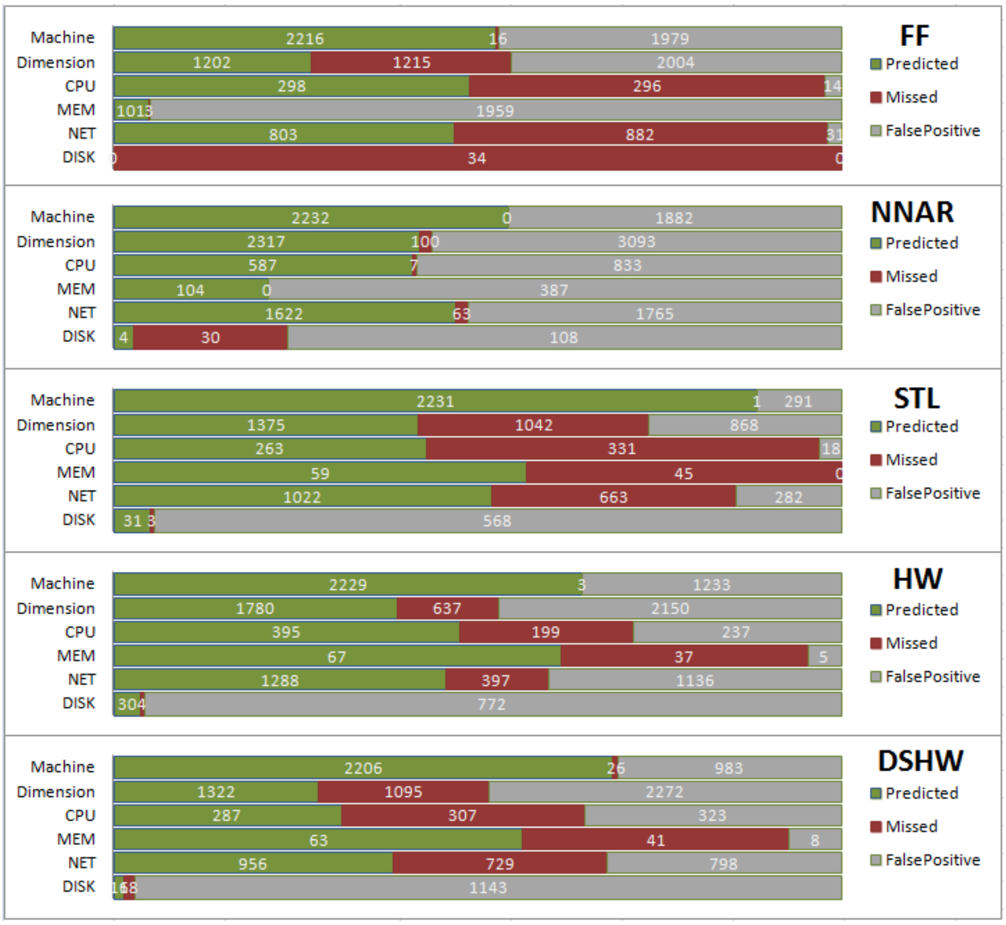


Figure 7.13: Simulation Results with learning applied on 3 days data and tested on the last day period. Confidence level is set to 90% and window size is 30 data points. FF: Fourier forecast; NNAR: Neural Network AutoRegression; STL: STL (Seasonal and Trend decomposition using Loess) based forecasting; HW: Holt Winter's exponential smoothing; DSHW: Double Seasonal Holt Winter's exponential smoothing

seasonal time series, such as our input. The first simulation results are shown in Figure 7.13, we used fixed confidence level for each dimension (90) and fixed lookup horizon (30).

In each simulation we used the same bin size: the same capacity dimensions for the simulated machines. Every resource dimension is limited in that way that it is likely to be violated during the simulation. The input is also the same for each case, and the migration rules are unchanged too. This results a reproducible comparison: in each simulation the machine violations are the same ones and the sum of them are also identical, we simply compare the forecasting methods, how many times the violation could have been predicted by the calculated forecasts.

For the comparison it is also crucial to know how many times certain forecasting methods and rules deliver false positive violation warnings. On one hand, no violation is desirable, and this results in a higher false positive rate. On the other hand, higher false positive rate means that the scheduler has to migrate the applications more often, and migration introduces additional cost to the cloud in general, which should also be avoided. The perfect simulation would have a low false positive rate but a high prediction accuracy. The number of false positive warnings is generally bad for overall utilization: in a real system the migration process would be initialized by a false positive violation prediction too. The violation is most likely to be avoided, but in case of a false positive, violation would not have happened on the original machine either. Unnecessary migration leaves the original machine less utilized and introduces additional cost.

So it can be seen that it is necessary to investigate not just the prediction accuracy but also the false positive rate. The false positives are compared to the worst case scenario. The worst case scenario is given by the MAX-forecast, which means in every simulation step for every application it is assumed that the resource needs in the next step will be equal to the historical maximum. This approach is going to not only correctly predict all violations, but also this approach results in a the highest false positive rate. We compare the rest of the false positive rates to this worst case scenario.

This test took 3 day application traces as inputs, the forecast methods are trained on all 3 days, the forecast horizon is one day. The simulation takes the real data of the last day. In other words the train set is the whole data set and the test set is the last day (last 1440 data points).

The best performing prediction is given by NNAR, which predicts all machine violations correctly in the simulation, and for each dimension it delivers highly accurate predictions. Although, there are also a lot of false positives.

The second place goes to the STL based forecast which only misses one violation, but on the plus side, there are much less false positives during the simulation. The per dimension forecast accuracy is worse. Note that if some dimension violation is predicted and another dimension violation happens during the simulation, that still counts as a correct prediction on the machine. This makes sense because the correlation between the resource usages are strong, and violation in one dimension is a strong predictor of a violation in another dimension.

Holt Winters, and Fourier are average performing, and Double Seasonal Holt Winters method performs the worst, although on NET and DISK dimensions better, than fourier.

Generally, we see that the prediction of the DISK dimension is the most inaccurate due to high variance, and the most stable prediction is for the memory usage.

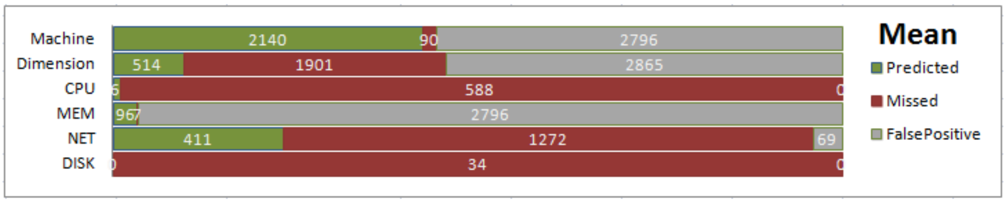


Figure 7.14: Prediction by forecasting the mean value of the traces, using 90% confidence level and 30 data points window size.

Figure 7.14 shows the results of the prediction when the mean of the historical traces are forecasted for the future. A naive approach would work by such simple rule, and as it can be concluded the overall prediction happens to be not as bad as expected. The reason is that on average, half of the application resource usages that are running on a machine will be overestimated and the other half underestimated, which together will give a quite good estimation. Looking at each dimension separately it is clear that this forecast method is a poor choice. Comparing the other methods to this simple approach, it can be seen that they all outperform basic mean forecasting.

Results with different settings

In this section the effect of using different confidence levels and different window sizes on the results is compared. This time we give the results as percentages. Percentage is calculated as follows:

$$Predicted(\%) = (NumPredicted/NumViolations) * 100,$$

$$Missed(\%) = (NumMissed/NumViolations) * 100,$$

$$FalsePositive(\%) = (NumFalsePositive/NumMaxFalsePositive) * 100.$$

Holt Winter's exponential smoothing with changing window size.

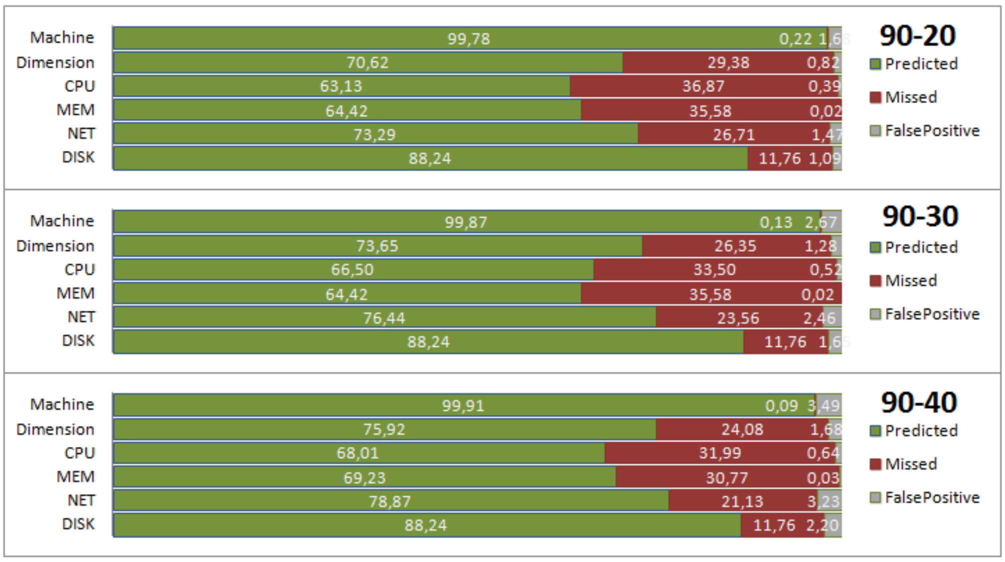


Figure 7.15: Holt Winters with changing window size: Window size is the second number in the chart title: 20,30,40. The confidence level is fixed at 90%

Holt Winter's exponential smoothing with changing confidence level. This test was done with all the other used forecasting methods, the results can be found in the Appendix C. From the results above it is clear that using higher confidence level results in more false positives and more misses, and narrowing down the lookup window will result a higher rate of missed predictions and less false positives.

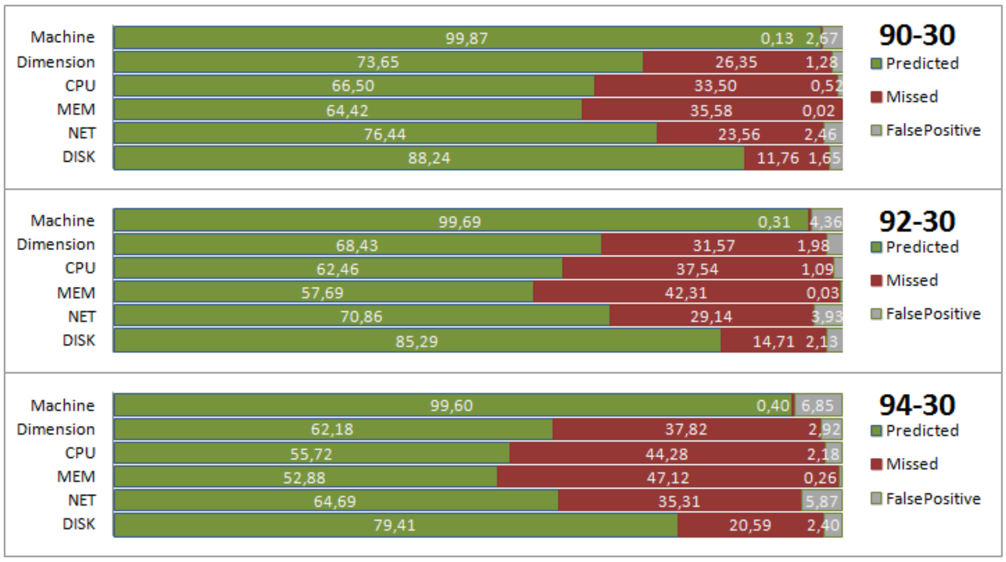


Figure 7.16: Holt Winters with changing confidence level: confidence level is the first number in the chart title: 90,92,94. The window size is fixed at 30 data points.

These settings are important to have since different cloud providers are willing to take different amounts of risk: some of them are willing to take high risk which means some SLA violations are going to slip trough and only handled by the reactional rules but the overall utilization of the cloud remains higher. Other cloud providers would rather have less maximized utilization, but no violation at all.

Confidence level and window size offer cloud administrators the opportunity to customize the level of risk they are willing to take in order to maximize the cloud utilization.

7.5.8 Feedback loop

The previous simulations were run by training the forecasting algorithms on the whole input data set (3 days) and test take the last day as the real usage traces (test set). However if we train the algorithms on just the first two days and use the third day as test set, the results are more inaccurate. The reason is that we simulated a level shift in the data: the algorithms should forecast a level change that does never happened before, this clearly can not be forecasted accurately. There are 3 options to overcome these inaccuracy:

- When the error rate surpass a certain threshold switch classification for the time series and use a different forecasting algorithm, for example trend estimation for shorter horizon.
- Switch to other seasonal forecast algorithms: STL and Fourier usually average the magnitude which more likely results better estimation than NNAR,HW, DSHW can provide.
- Using error adjustment, which we explain in this section.

These types of errors are the reason that such forecasting system should rather focus on dynamically switching the forecasting method, reclassify, correct errors rather than find one universal forecasting algorithm.

During the simulation, just like in real system the difference between the actual usage and the forecasted usage for each resource, in every time interval can be inspected in real time. Using this, different types of error correction adjustment can be applied. We call this error correction a feedback loop. The real time feedback of errors are used to correct the forecast values. With our simulation we implemented a simple case of error correction adjustment: in every lookup window the average difference between the actual usage and the forecasted usage is calculated and this value is added or subtracted automatically to/from the forecast values of next window.

$$correction = \frac{\sum_{i}^{i+windowSize}(actualValue_i - forecastedValue_i)}{windowSize}$$

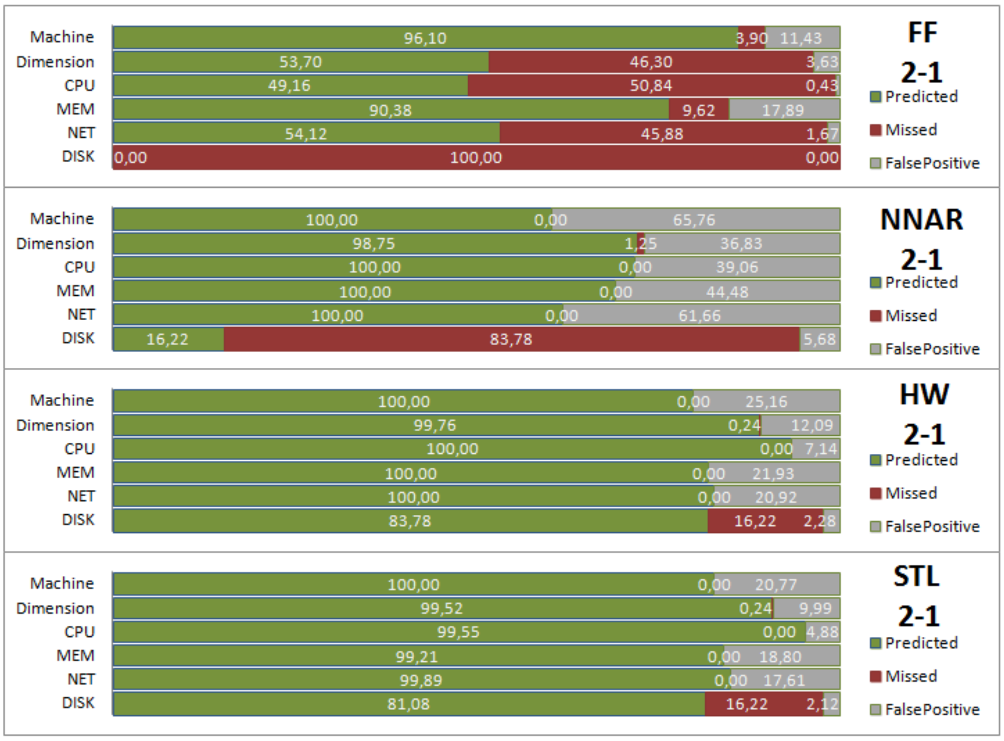


Figure 7.17: Results on 2days - 1 day split. Confidence level is set to 90% and window size is 30 data points. FF: Fourier forecast; NNAR: Neural Network AutoRegression; STL: STL (Seasonal and Trend decomposition using Loess) based forecasting; HW: Holt Winter's exponential smoothing; DSHW: Double Seasonal Holt Winter's exponential smoothing

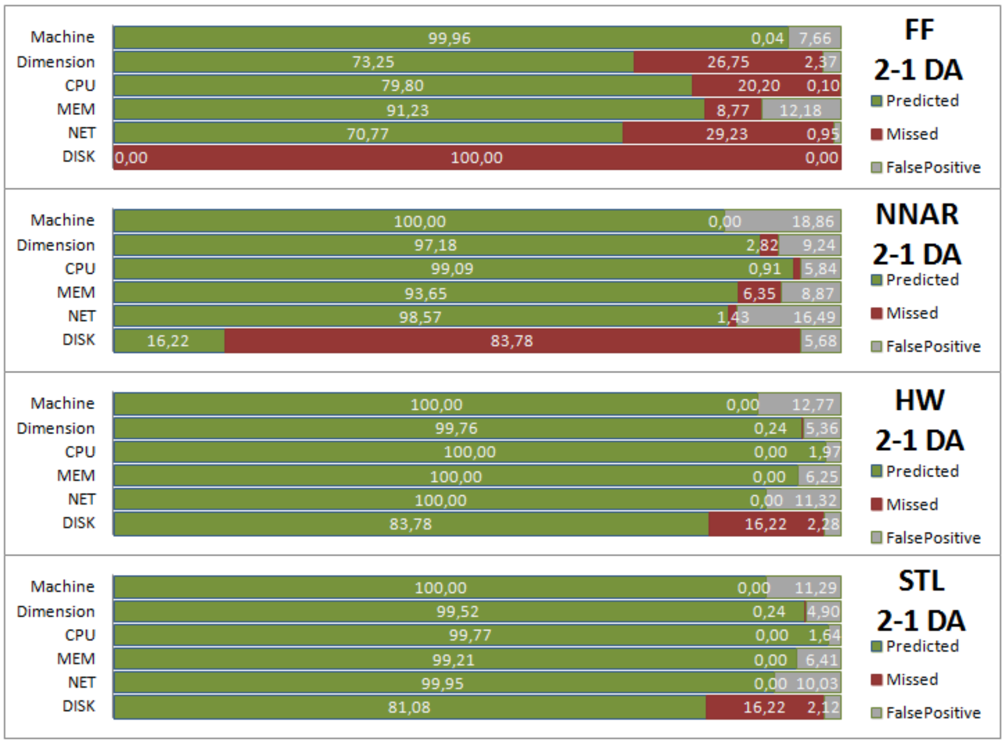


Figure 7.18: Results on 2days - 1 day split with dynamic error correction. Confidence level is set to 90% and window size is 30 data points. FF: Fourier forecast; NNAR: Neural Network AutoRegression; STL: STL (Seasonal and Trend decomposition using Loess) based forecasting; HW: Holt Winter's exponential smoothing; DSHW: Double Seasonal Holt Winter's exponential smoothing

The first Figure (7.17) shows the results after training the forecasting methods on just the first 2 days and testing it on the third day without using error correction. A large proportion of false positives and in case of fourier forecast the missed prediction percentage means that the forecasts are mostly inaccurate, they miss the level shift. In this scenario STL based forecast performs best. The reason is that it averages the previous magnitudes for the next forecast period. Fourier forecast tends to underestimate the resource usages, and results in higher missed percentage on each resource dimension. NNAR and HW both suffer from overestimation errors (especially NNAR). This can be seen from the high percentage of false positive warnings.

The second Figure (7.18) shows the results after applying the error correction. As the result of the error correction false positive rates are decreased dramatically while preserving the prediction accuracy. STL decomposition based forecasting outperforms the other methods by showing high prediction accuracy and low false positive rates.

This time the scenario is backtested with comparing it to the naïve forecasting with applied dynamic error correction. This forecast works by copying the previous period exactly, see Figure 7.19.

This approach is outperformed by every other statistical forecasting method applied earlier.

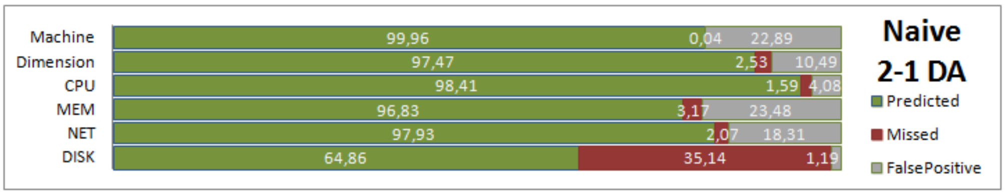


Figure 7.19: Naïve forecast with dynamic error correction, applied for 2 days - 1 day splitted data set. Confidence level is set to 90% and window size is 30 data points.

The dynamic error correction is a powerful technique to increase prediction accuracy. The results suggest (low missed dimensions and high false positive rates) that one could decrease the lookup window and still have an accurate prediction on violations, but in order to keep the results comparable the window size remains fixed during the experiments.

In both scenarios (2 days - 1 day and 3 days - 1 day split) the STL and HW methods turned out to be the best performing on forecasting the CPU usage. They have the lowest missed predictions and lowest false positives. Note that the table 5.3 shows the order in which the different forecasting methods are chosen by the classification process, when the observed trace has periodical fluctuations (seasonality). The ranking depends on which accuracy metric the classification process compares. On the first place according to every accuracy metric the DSHW method is chosen. But the extremely low values suggest not to use DSHW because it is the classic case of overfitting. This is proven by the simulation results where DSHW method has equally bad performance on predicting the CPU usage as fourier forecast.

On the second and third place STL and HW methods are ranked. The correct order (1.STL, 2. HW) is only given when ME (Mean Error) and MPE (Mean Percentage Error) metrics are used as base of the comparison.

CHAPTER 8

Summary and Future Work

This thesis describes the theoretical architecture of an adaptive and scalable proactive scheduling system, that monitors the cloud state in real time, applies statistical forecasting methods to extrapolate the resource usages and tries to adapt to future cloud state in advance. The domain of statistical forecasting was researched in depth and many of the available methods were described, that can be used in this context. Since there are many possible algorithms to be used for forecasting, we described a possible classification process to select one of these method that is best applicable for a specific scenario: The goal is to deliver the most accurate forecast based on arbitrary selected accuracy measure and the presence of various time series features such as trend, seasonality and variance. We also provided in-depth description of the advantages and disadvantages of these accuracy metrics and the detection of trend, seasonality and high variance.

For the tests we needed to generate test data for which we used JMeter as described in chapter 4. Additionally, we compared the available workload generation tools.

A cloud simulator was implemented that uses the generated data set to validate the idea of proactive violation detection based on the seasonal forecasting methods. The simulator provides an easy way to visually compare the forecasting methods. Using the simulator we benchmarked seasonal forecasting methods such as NNAR, Holt winters exponential smoothing, STL decomposition based forecasting, and Fourier forecasting, how accurately they can predict SLA violations that arise on a simulated machine entity.

On top of the seasonal forecasting we experienced successfully with dynamic error correction mechanisms to enhance prediction accuracy.

It is important not just to detect an arising violation, but to react on it and to find an optimal target machine to migrate the application onto. Described were the basic heuristical approaches to be used: one of them is describing how the observed seasonal/periodical behavior of applications resource usages can be used to enhance existing migration target selection.

Since the calculations require significant system resources themselves, dimensionality reduction techniques were presented to ease the calculations. During dimensionality reduction some detail of data are lost but the calculations are significantly faster.

To test the whole process it is important to have different input data. This either comes from real cloud infrastructure, or from generation of diverse workload patterns. LIMBO [55] [73] is an eclipse plugin developed by Kalrsruhe Institute of Technology which allows the modeling and generation of diverse statistical load intensity. It is developed to create diverse user request patterns over time so that different statistical forecasting methods can be tested on high variety of generated patterns. JMeter can be extended that it uses the LIMBO generated request pattern to create dynamically user threads that access to the workload while monitoring the VM resource usage. This way one would be able to generate more sophisticated input traces for the simulation.

This thesis provides the theoretical basics of the classification process. The implementation and testing such classification remains future work. It is strongly believed that the key of a successful forecasting system is the adaptive reclassification of the time series.

The simulator is essential to test different forecast methods, approaches, error correction methods, migration heuristics, etc for cloud scheduling systems.

There are certain additional possible extensions to make the simulation more realistic in the future:

- using variable bin sizes (physical computers across the cloud most likely do not have the same capacity)
- simulate the virtual machines (in the simulator now only machine is simulated as an entity because of the assumption that only the turning up and down of the physical hardware matters from the aspect of electricity usage)
- upper and lower bound should be configurable separately, and also for each different resource dimension there should be different lower and upper bound configurable.

We measure prediction, missed and false positive warning percentages. This could be extended with the actual implementation of the violation detection rule, implementation of different migration heuristics, and simulation of the actual migration while monitoring the overall cloud utilization: how many machines are in use and what percentage of the different resources are in use. With that implemented, one is able to directly check the effect of the different forecasting methods, violation detection rules, migration heuristics on the utilization.

Adaptive confidence level, lookup window: Adaptive error correction should be extended that based on the inspected errors, missed violations, not just the error is corrected but the confidence level, and the lookup window is dynamically adjusted.

The applicability of the classification we proposed has to be proven by simulations on different traces, with different configurations. The list of the implemented forecasting tools has to be extended.

CloudSim [18] project has many aspects of realistic cloud simulator already implemented such as different allocation policies, simulation dynamic insertion of elements, cloud federation, energy aware computing resources simulation. In a long term an integration of our forecasting based scheduling in the CloudSim project environment would enable to test many aspects of forecasting based scheduling.

The spectrum of the forecasting methods used and introduced in this thesis is not complete. The recent research of Professor Hyndmann seems promising and in the future his methods should be tested for resource forecasting of cloud computing systems. Since each resource usage metric are expected to correlate, using grouped forecasting method presented in [38] may give good results. Also there is maybe more complex seasonal periods to investigate in the traces, to forecast complex seasonal patterns we introduced the TBATs method, although the implementation and testing of the method is not included in this thesis [21].

APPENDIX A

Observed Workload Traces

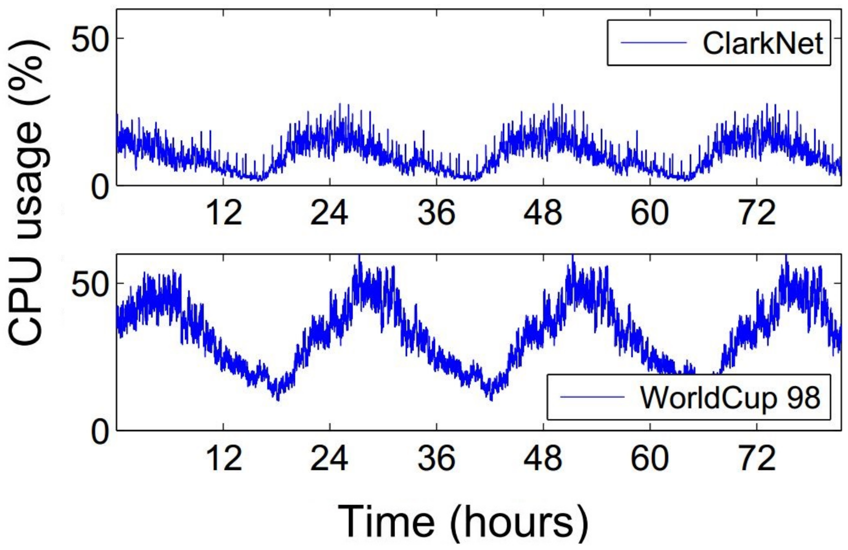


Figure A.1: Real seasonal traces from ClarkNet, and The number of accesses during the world-cup ’98 [34].

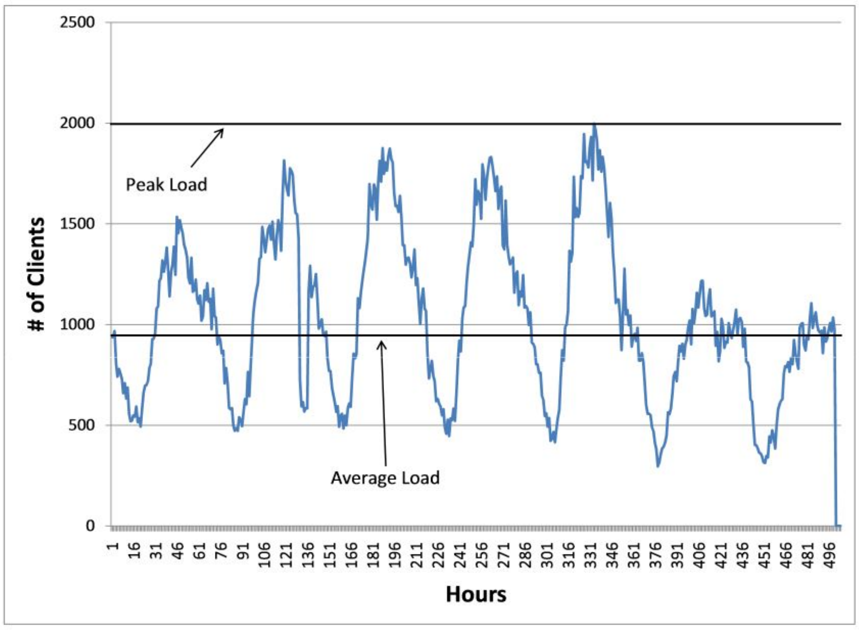


Figure A.2: World Cup 98 Traces on a longer scale (one week) [64].

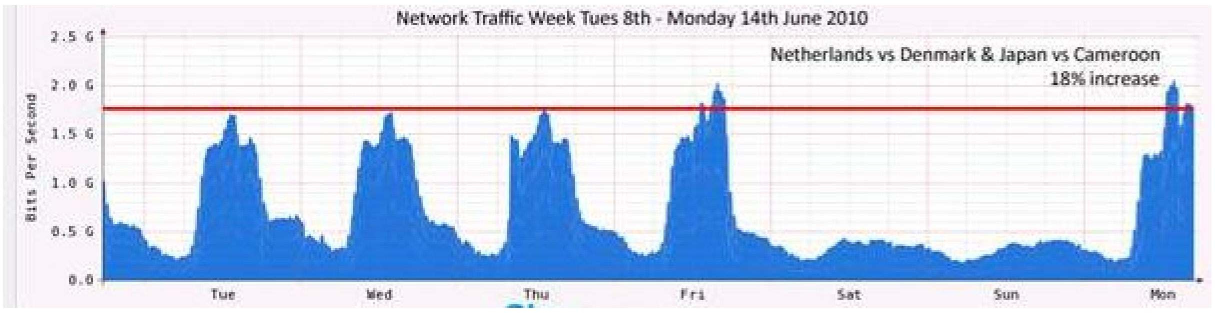


Figure A.3: Network statistics from Star's (UK business ISP and on-demand computing supplier) data centre also showed an increase on normal traffic level as workers followed their teams online. The graph below highlights that network traffic during the South Africa vs Mexico opening match and Holland vs Denmark and Japan vs Cameroon matches showed two network traffic peaks of up to 18% above normal [67].

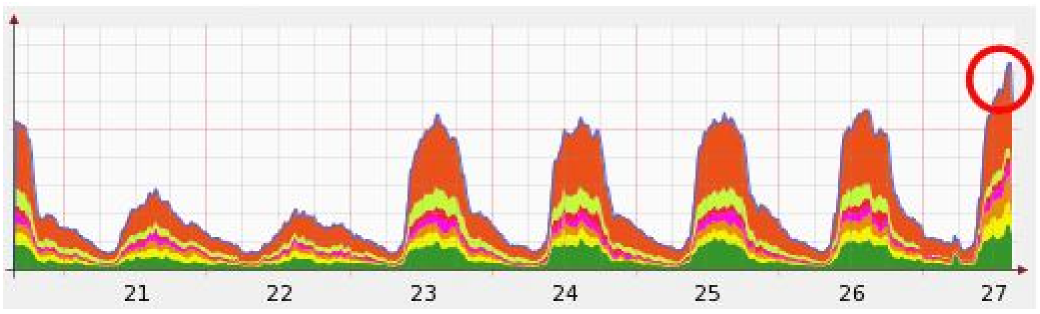


Figure A.4: Internet traffic from 2012 Olympic Games in London. Peak in the red circle is at the time of the tennis match between Djokovic and Murray (27th Jan 2012) [26]

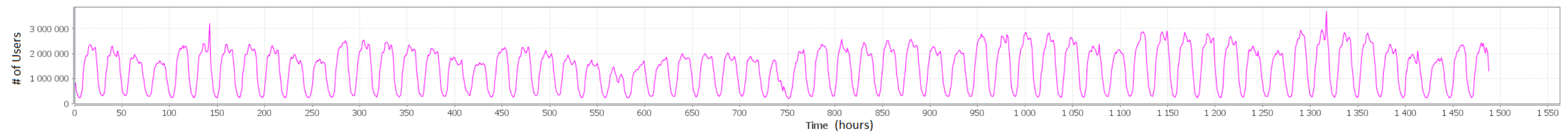


Figure A.5: Number of users accessed to the wikipedia's germany article per hour, as extracted from the wikipedia traffic collection: [51].

APPENDIX B

Time Series Analysis

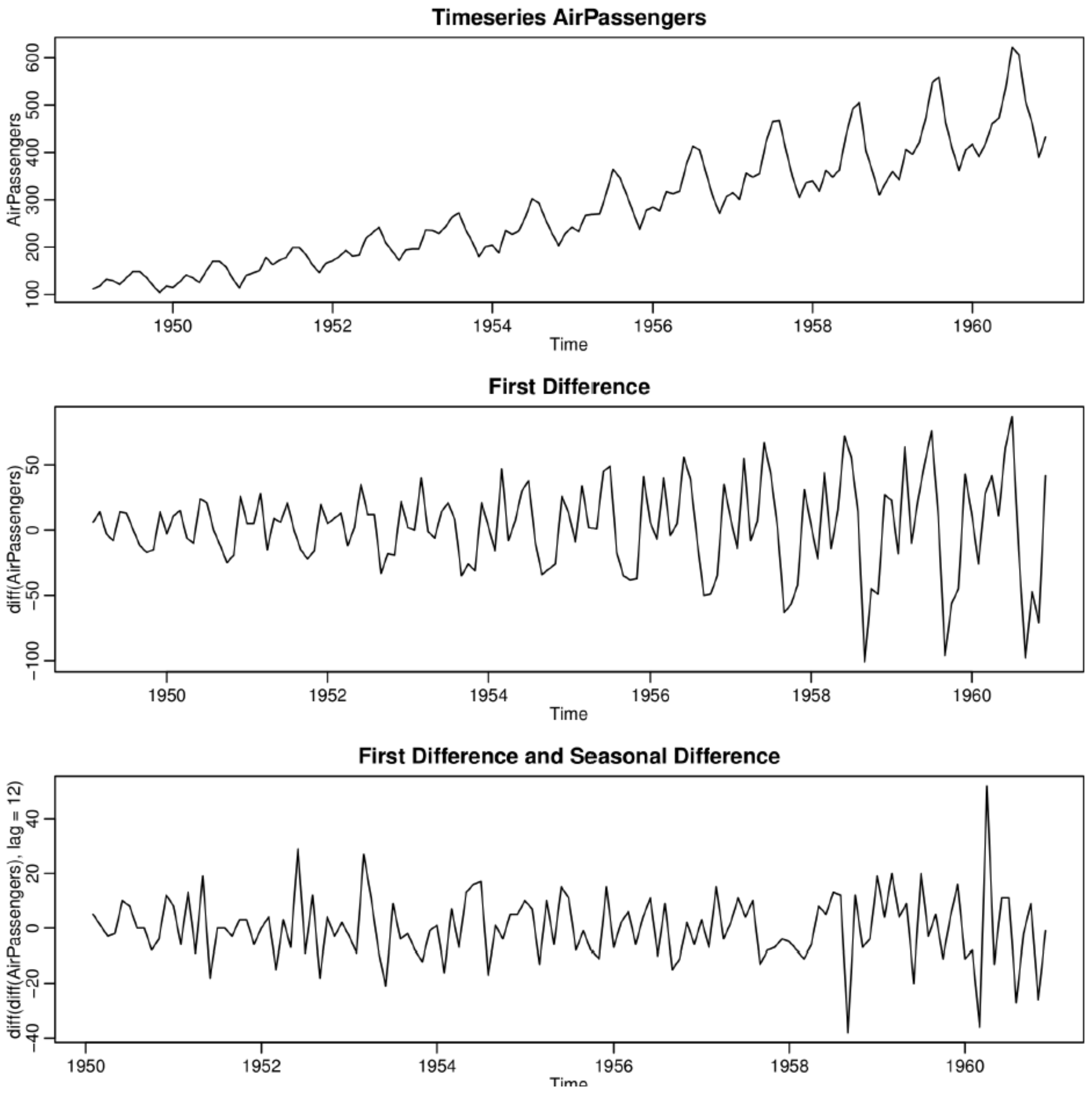


Figure B.1: Air Passengers Time Series Plots. The first chart shows the plot of the raw data, the second shows the first difference on lag 1, and the third is additionally differenced by the seasonal lag, which is 12 [11].

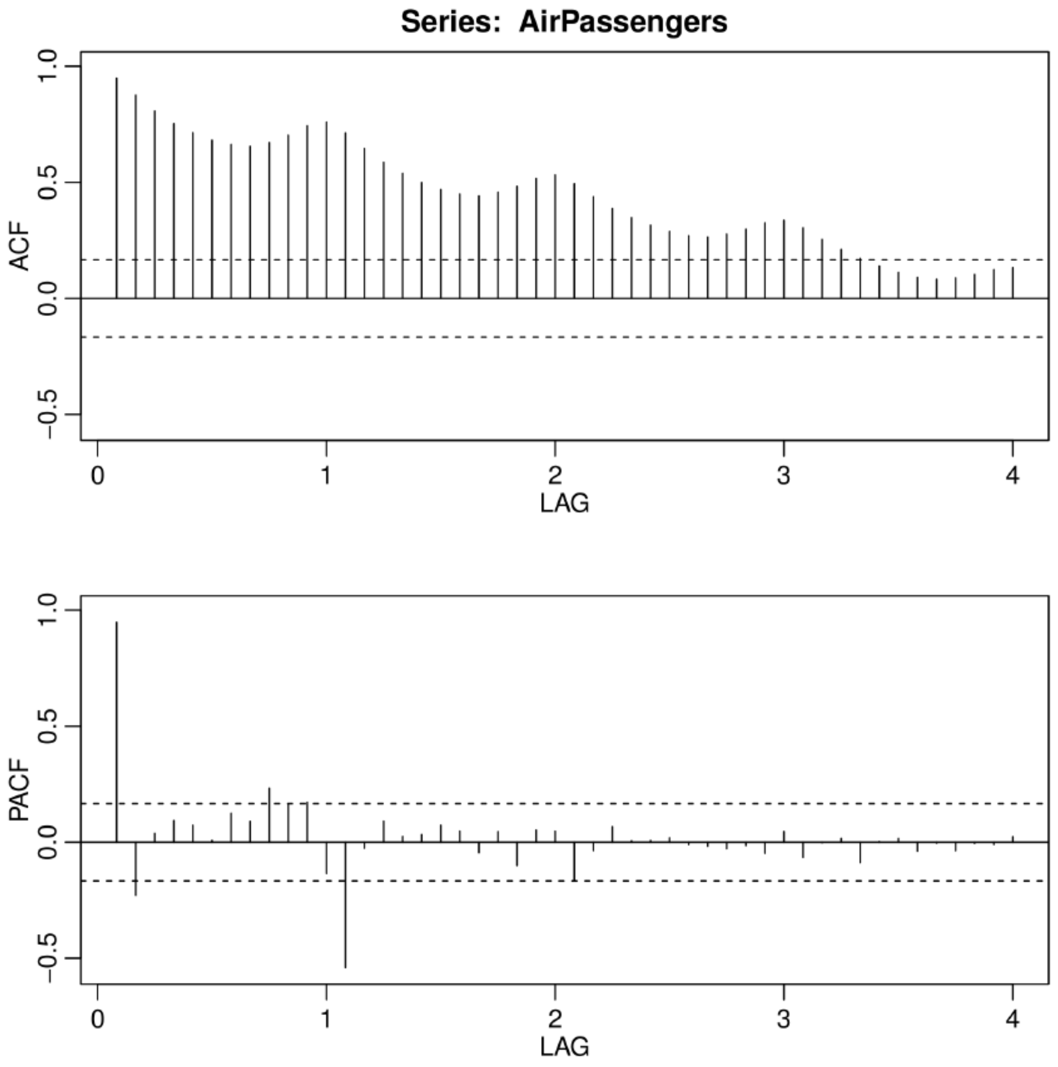


Figure B.2: ACF PACF applied on the Air Passengers Time Series data [11].

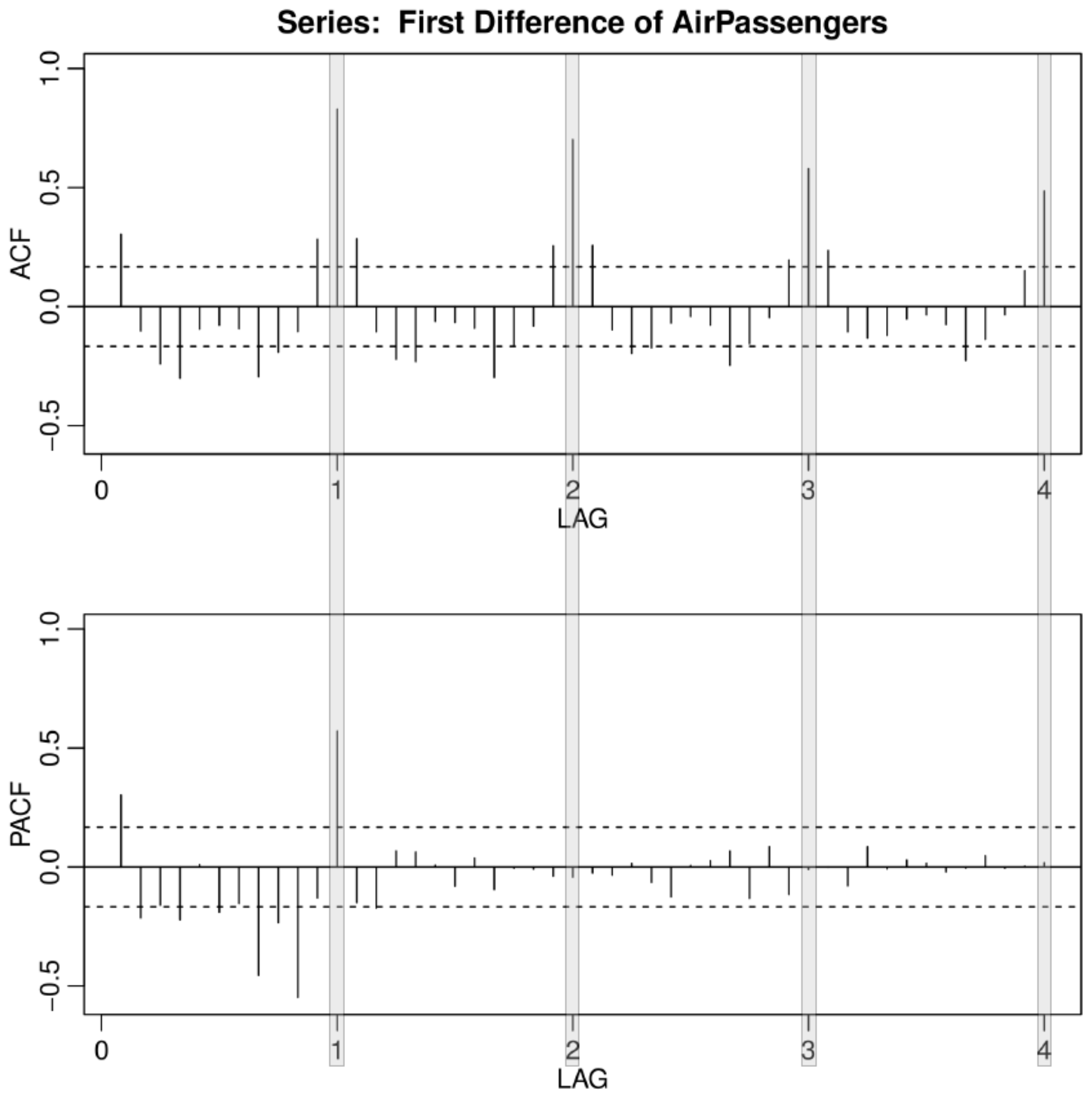


Figure B.3: ACF PACF applied on the differenciated Air Passengers Time Series data [11].

APPENDIX C

Simulation Results

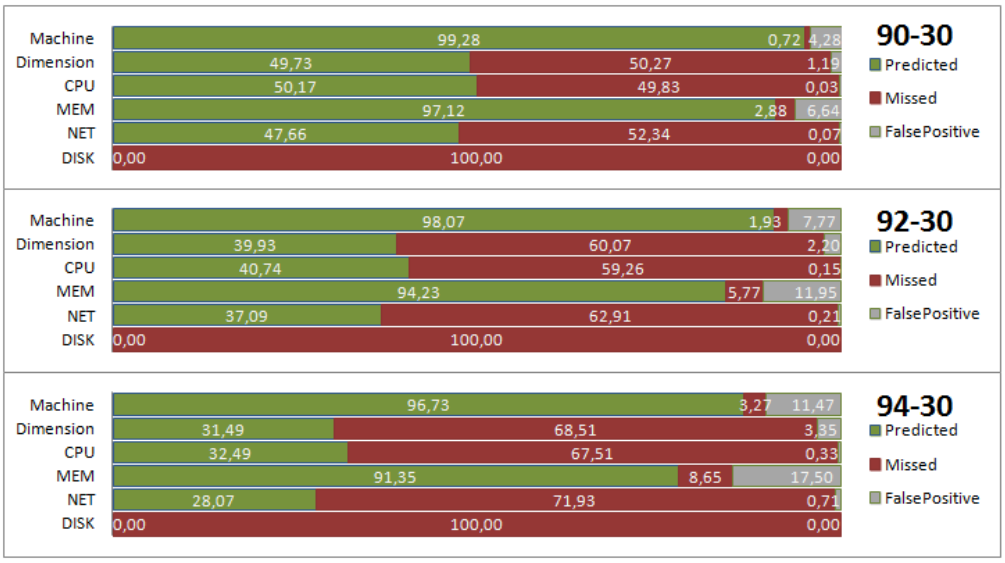


Figure C.1: Fourier forecast with changing window size. The first number in the chart title is the confidence level, the second is the window size.

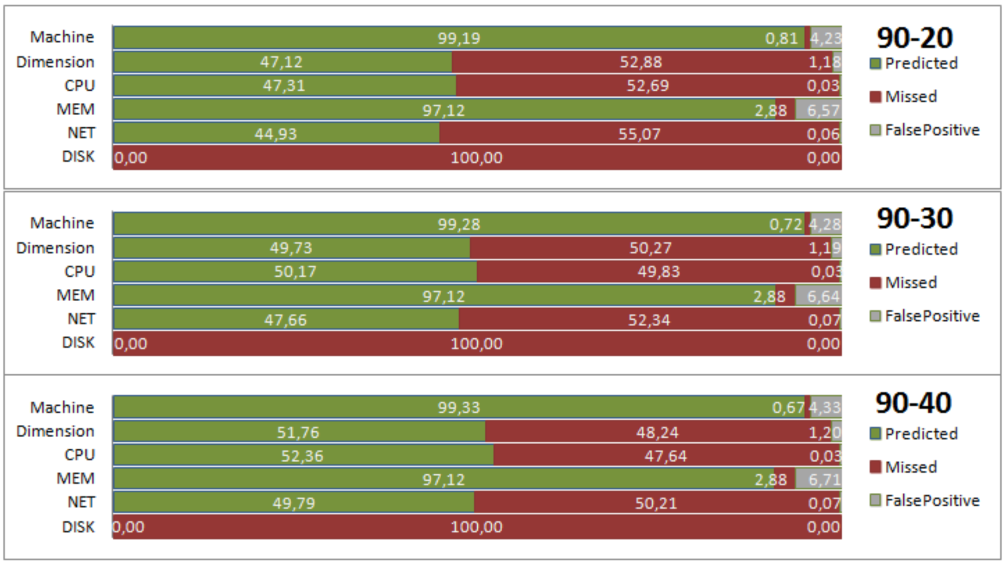


Figure C.2: Fourier forecast with changing confidence level. The first number in the chart title is the confidence level, the second is the window size.

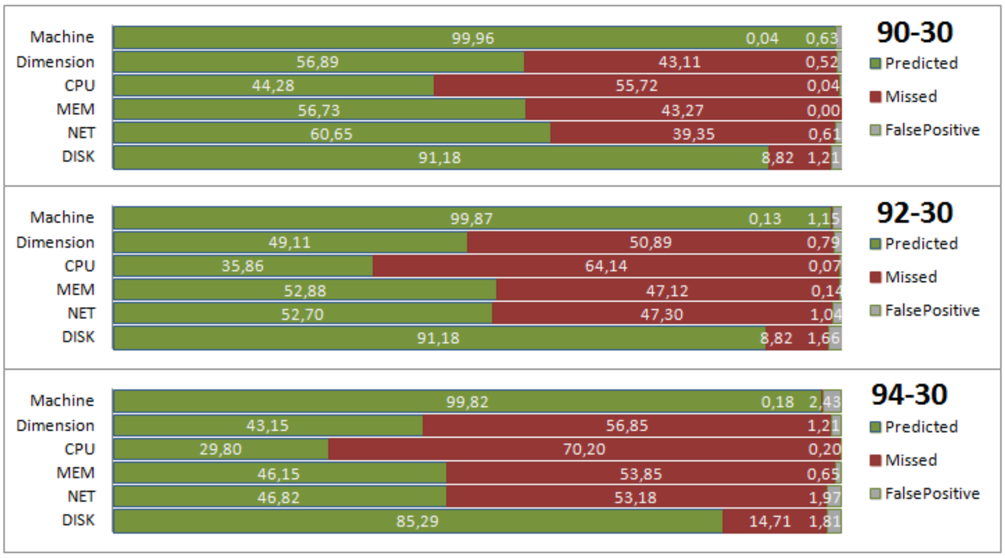


Figure C.3: STL forecast with changing window size. The first number in the chart title is the confidence level, the second is the window size.

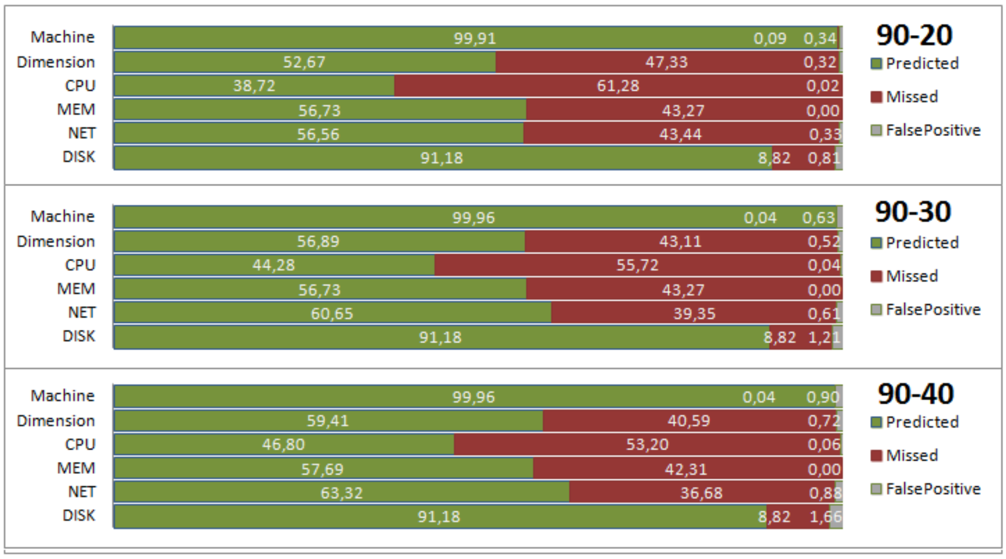


Figure C.4: STL forecast with changing confidence level. The first number in the chart title is the confidence level, the second is the window size.

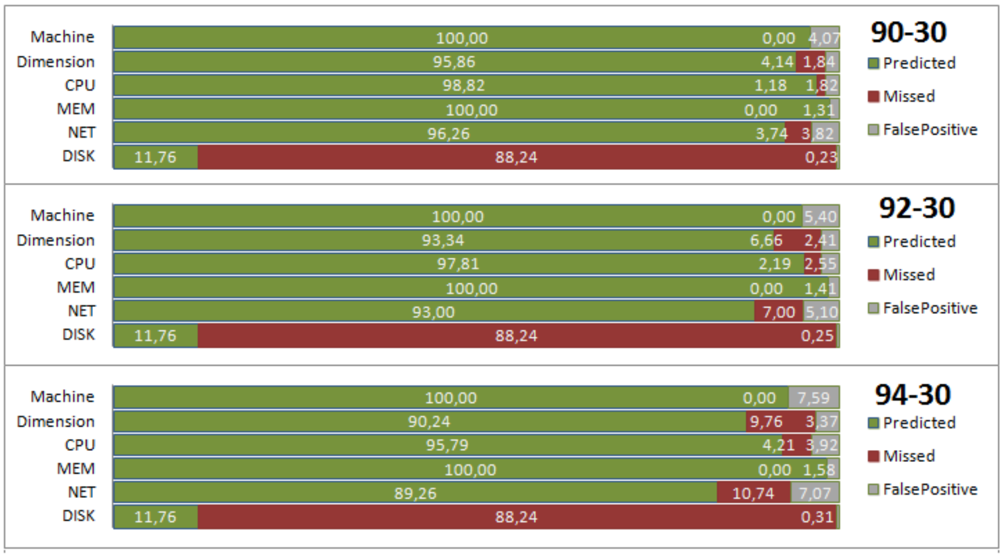


Figure C.5: NNAR (Neural Network Autoregression) with changing window size. The first number in the chart title is the confidence level, the second is the window size.

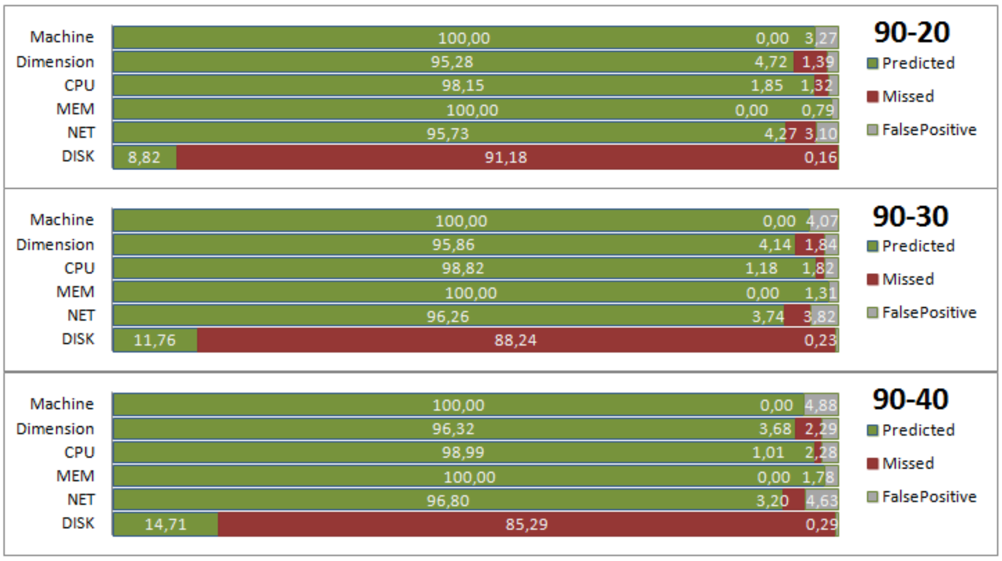


Figure C.6: NNAR (Neural Network Autoregression) with changing confidence level. The first number in the chart title is the confidence level, the second is the window size.

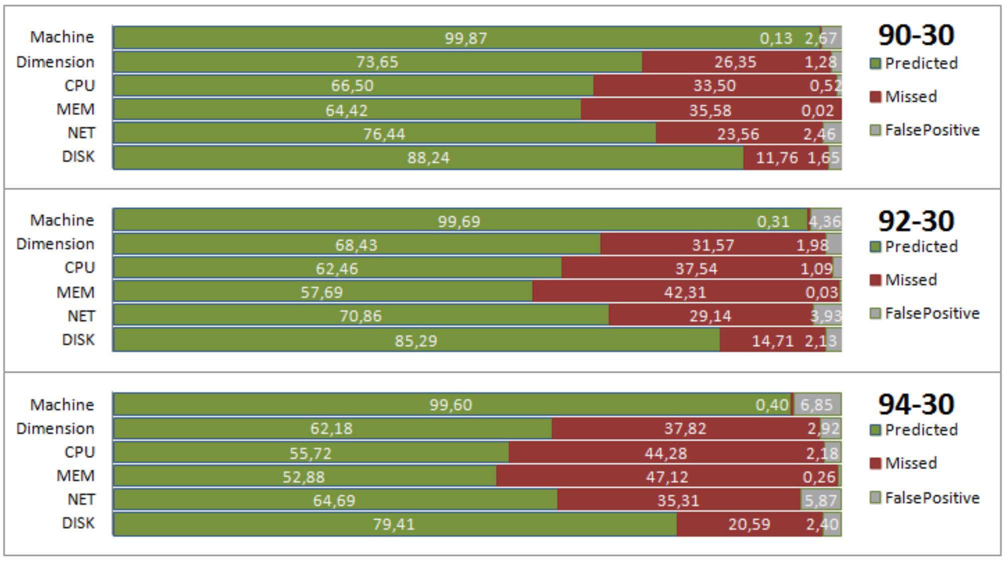


Figure C.7: Holt Winters with changing window size. The first number in the chart title is the confidence level, the second is the window size.

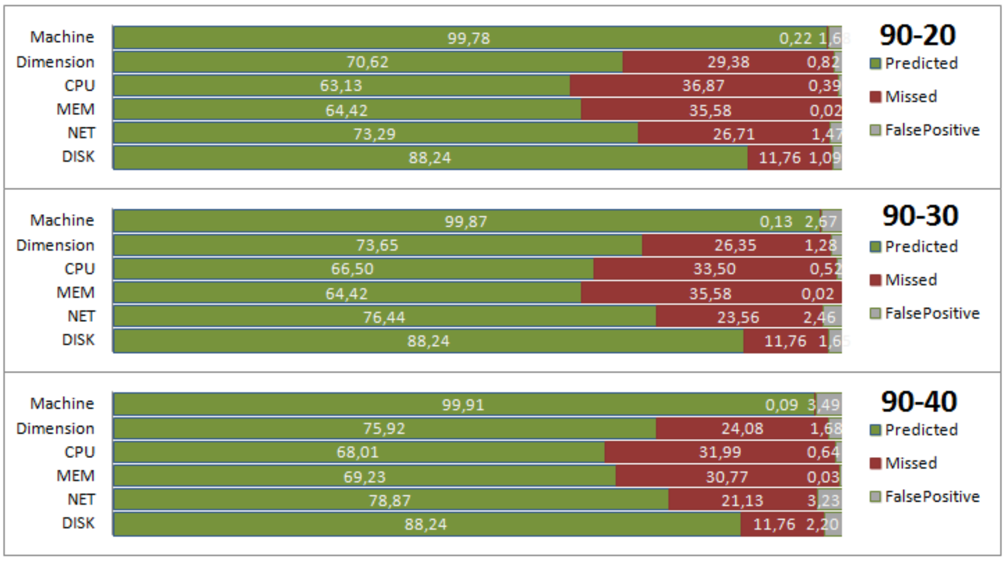


Figure C.8: Holt Winters with changing confidence level. The first number in the chart title is the confidence level, the second is the window size.

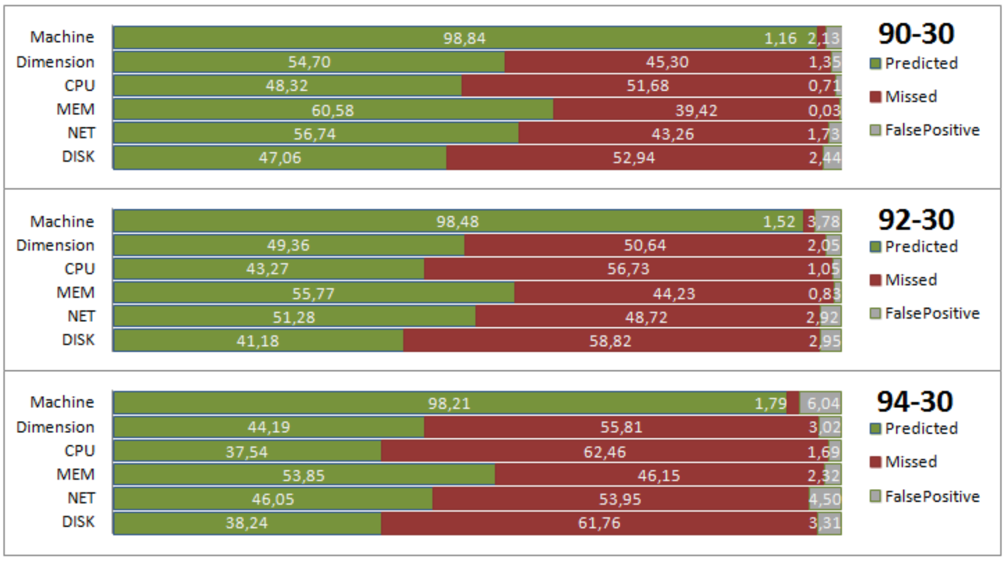


Figure C.9: Double seasonal Holt Winters with changing window size. The first number in the chart title is the confidence level, the second is the window size.

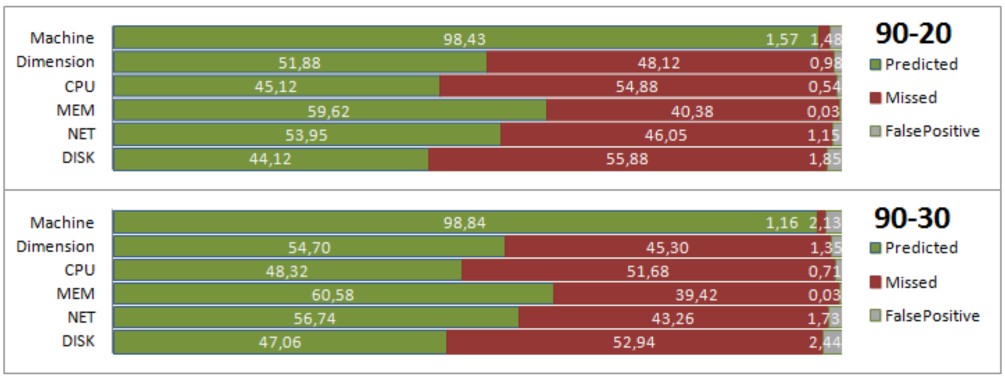


Figure C.10: Double seasonal Holt Winters with changing confidence level. The first number in the chart title is the confidence level, the second is the window size.

Bibliography

[1] Ratnadip Adhikari and R. K. Agrawal. An introductory study on time series modeling and forecasting. *CoRR*, abs/1302.6613, 2013.

[2] Nesreen K. Ahmed, Amir F. Atiya, Neamat El Gayar, and Hisham El-shishiny. An empirical comparison of machine learning models for time series forecasting, 2010.

[3] Amazon. Amazon elastic cloud. http://aws.amazon.com/autoscaling/. 2014-03-26.

[4] Laura Scarabotti Andrea Fumi, Arianna Pepe and Massimiliano M. Schiraldi. Fourier analysis for demand forecasting in a fashion company. *International Journal of Engineering Business Management*, 2013.

[5] Mauro Andreolini and Sara Casolari. Load prediction models in web-based systems. In *Proceedings of the 1st International Conference on Performance Evaluation Methodolgies and Tools*, valuetools '06, New York, NY, USA, 2006. ACM.

[6] M.F. Arlitt and C.L. Williamson. Internet web servers: workload characterization and performance implications. *Networking, IEEE/ACM Transactions on*, 5(5):631–645, Oct 1997.

[7] Azurewatch. Azurewatch. http://www.paraleap.com/azurewatch. 2014-03-26.

[8] L.A. Barroso and U. Holzle. The case for energy-proportional computing. *Computer*, 40(12):33–37, Dec 2007.

[9] Aaron Beitch, Brandon Liu, Timothy Yung, Rean Griffith, Armando Fox, and David A. Patterson. Rain: A workload generation toolkit for cloud computing applications. Technical Report UCB/EECS-2010-14, EECS Department, University of California, Berkeley, Feb 2010.

[10] Anton Beloglazov, Rajkumar Buyya, Young Choon Lee, and Albert Zomaya. A taxonomy and survey of energy-efficient data centers and cloud computing systems. Technical Report CLOUDS-TR-2010-3, Cloud Computing and Distributed Systems Laboratory, The University of Melbourne, august 2010.

[11] Markus Boegl. Visual analytics for time series analysis. Master's thesis, Technische Universität Wien, Austria, 2013.

[12] G. E. P. Box and N. R. Draper. *Empirical Model-Building and Response Surfaces*. John Wiley, New York, NY, 1987.

[13] George Edward Pelham Box and Gwilym Jenkins. *Time Series Analysis, Forecasting and Control*. Holden-Day, Incorporated, 1990.

[14] Roy Bryant, Alexey Tumanov, Olga Irzak, Adin Scannell, Kaustubh Joshi, Matti Hiltunen, Andres Lagar-Cavilla, and Eyal de Lara. Kaleidoscope: Cloud micro-elasticity via vm state coloring. In *Proceedings of the Sixth Conference on Computer Systems*, EuroSys '11, pages 273–286, New York, NY, USA, 2011. ACM.

[15] K.P. Burnham and D.R. Anderson. *Model Selection and Multimodel Inference: A Practical Information-Theoretic Approach*. Springer, 2002.

[16] Emmanuel Cecchet, Veena Udayabhanu, Timothy Wood, and Prashant Shenoy. Benchlab: An open testbed for realistic benchmarking of web applications. In *Proceedings of the 2Nd USENIX Conference on Web Application Development*, WebApps'11, pages 4–4, Berkeley, CA, USA, 2011. USENIX Association.

[17] Robert B. Cleveland, William S. Cleveland, Jean E. McRae, and Irma Terpenning. Stl: A seasonal-trend decomposition procedure based on loess (with discussion). *Journal of Official Statistics*, 6:3–73, 1990.

[18] The Cloud Computing and University of Melbourne Distributed Systems (CLOUDS) Laboratory. Cloudsim: A framework for modeling and simulation of cloud computing infrastructures and services. http://www.cloudbus.org/cloudsim/. 2014-04-13.

[19] Transaction Processing Performance Council. Tpc-w benchmark. http://www.tpc.org/tpcw/. 2014-03-06.

[20] J.D. Cryer and K.S. Chan. *Time Series Analysis: With Applications in R*. Springer Texts in Statistics. Springer, 2008.

[21] Alysha M. De Livera, Rob J. Hyndman, and Ralph D. Snyder. Forecasting time series with complex seasonal patterns using exponential smoothing. *Journal of the American Statistical Association*, 106(496):1513–1527, 2011.

[22] Anh Vu Do, Junliang Chen, Chen Wang, Young Choon Lee, A.Y. Zomaya, and Bing Bing Zhou. Profiling applications for virtual machine placement in clouds. In *Cloud Computing (CLOUD), 2011 IEEE International Conference on*, pages 660–667, July 2011.

[23] Prince Jain Dr. Rahul Malhotra. Study and comparison of various cloud simulators available in the cloud computing. *International Journal of Advanced Research in Computer Science and Software Engineering*, 3, 2013.

[24] Philippe Esling and Carlos Agon. Time-series data mining. *ACM Comput. Surv.*, 45(1):12, 2012.

[25] Xiaobo Fan, Wolf-Dietrich Weber, and Luiz Andre Barroso. Power provisioning for a warehouse-sized computer. In *Proceedings of the 34th Annual International Symposium on Computer Architecture*, ISCA '07, pages 13–23, New York, NY, USA, 2007. ACM.

[26] UK ISP Fluidata. Isp data during 2012 olympics. http://www.ispreview.co.uk/story/2012/01/30/uk-isp-fluidata-warns-office-internet-traffic-could-be-hit-hard-by-olympic-games.html. 2014-04-22.

[27] The Apache Software Foundation. Jmeter. https://jmeter.apache.org/. 2014-03-08.

[28] The Apache Software Foundation. Jmeter plugins. http://jmeter-plugins.org/. 2014-03-08.

[29] A. Gandhi, Yuan Chen, D. Gmach, M. Arlitt, and M. Marwah. Minimizing data center sla violations and power consumption via hybrid resource provisioning. In *Proceedings of the 2011 International Green Computing Conference and Workshops*, IGCC '11, pages 1–8, Washington, DC, USA, 2011. IEEE Computer Society.

[30] Anshul Gandhi, Mor Harchol-Balter, Ram Raghunathan, and Michael A. Kozuch. Autoscale: Dynamic, robust capacity management for multi-tier data centers. *ACM Trans. Comput. Syst.*, 30(4):14:1–14:26, November 2012.

[31] Daniel F. Garcia and Javier Garcia. Tpc-w e-commerce benchmark evaluation. *Computer*, 36(2):42–48, 2003.

[32] Saurabh Kumar Garg, Chee Shin Yeo, Arun Anandasivam, and Rajkumar Buyya. Energy-efficient scheduling of hpc applications in cloud computing environments. *CoRR*, abs/0909.1146, 2009.

[33] D. Gmach, J. Rolia, L. Cherkasova, and A. Kemper. Capacity management and demand prediction for next generation data centers. In *Web Services, 2007. ICWS 2007. IEEE International Conference on*, pages 43–50, July 2007.

[34] Zhenhuan Gong, Xiaohui Gu, and John Wilkes. Press: Predictive elastic resource scaling for cloud systems. In *CNSM*, pages 9–16. IEEE, 2010.

[35] J. Octavio Gutiérrez-García and Kwang Mong Sim. A family of heuristics for agent-based elastic cloud bag-of-tasks concurrent scheduling. *Future Generation Comp. Syst.*, 29(7):1682–1699, 2013.

[36] Nikolas Roman Herbst, Nikolaus Huber, Samuel Kounev, and Erich Amrehn. Self-adaptive workload classification and forecasting for proactive resource provisioning. In *Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering*, ICPE '13, pages 187–198, New York, NY, USA, 2013. ACM.

[37] André Van Hoorn, Matthias Rohr, and Wilhelm Hasselbring. Generating probabilistic and intensity-varying workload for web-based software systems. In *Performance Evaluation – Metrics, Models and Benchmarks: Proceedings of the SPEC International Performance Evaluation Workshop (SIPEW '08), volume 5119 of Lecture Notes in Computer Science (LNCS*, pages 124–143. SPEC, Springer. ISBN, 2008.

[38] Rob J Hyndman, George Athanasopoulos, and Han Lin Shang. hts: An r package for forecasting hierarchical or grouped time series, 2013.

[39] Rob J Hyndman and Yeasmin Kh. Automatic time series forecasting: The forecast package for r. *Journal of Statistical Software*, 2008.

[40] Rob J Hyndman and Anne B Koehler. Another look at measures of forecast accuracy. *International Journal of Forecasting*, pages 679–688, 2006.

[41] Nima Kaviani, Eric Wohlstadter, and Rodger Lea. Profiling-as-a-service: Adaptive scalable resource profiling for the cloud in the cloud. In *Proceedings of the 9th International Conference on Service-Oriented Computing*, ICSOC'11, pages 157–171, Berlin, Heidelberg, 2011. Springer-Verlag.

[42] Eamonn Keogh, Kaushik Chakrabarti, Michael Pazzani, and Sharad Mehrotra. Dimensionality reduction for fast similarity search in large time series databases. *JOURNAL OF KNOWLEDGE AND INFORMATION SYSTEMS*, 3:263–286, 2000.

[43] Emre Kiciman and Benjamin Livshits. Ajaxscope: A platform for remotely monitoring the client-side behavior of web 2.0 applications. *SIGOPS Oper. Syst. Rev.*, 41(6):17–30, October 2007.

[44] J. Kolodziej, S.U. Khan, and F. Xhafa. Genetic algorithms for energy-aware scheduling in computational grids. In *P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC), 2011 International Conference on*, pages 17–24, Oct 2011.

[45] Harold C. Lim, Shivnath Babu, and Jeffrey S. Chase. Automated control for elastic storage. In *Proceedings of the 7th International Conference on Autonomic Computing*, ICAC '10, pages 1–10, New York, NY, USA, 2010. ACM.

[46] Object Refinery Limited. Jfreechart. http://www.jfree.org/jfreechart/. 2014-03-30.

[47] Jessica Lin and Yuan Li. Finding structural similarity in time series data using bag-of-patterns representation. In Marianne Winslett, editor, *Scientific and Statistical Database Management*, volume 5566 of *Lecture Notes in Computer Science*, pages 461–477. Springer Berlin Heidelberg, 2009.

[48] Chi-Keung Luk, Robert Cohn, Robert Muth, Harish Patil, Artur Klauser, Geoff Lowney, Steven Wallace, Vijay Janapa Reddi, and Kim Hazelwood. Pin: Building customized program analysis tools with dynamic instrumentation. In *Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '05, pages 190–200, New York, NY, USA, 2005. ACM.

[49] Oded Maimon and Lior Rokach, editors. *Data Mining and Knowledge Discovery Handbook, 2nd ed.* Springer, 2010.

[50] Toni Mastelic, Vincent C. Emeakaroha, Michael Maurer, and Ivona Brandic. M4cloud - generic application level monitoring for resource-shared cloud environments. In Frank Leymann, Ivan Ivanov, Marten van Sinderen, and Tony Shan, editors, *CLOSER*, pages 522–532. SciTePress, 2012.

[51] Domas Mituzas. Wikipedia traffic. http://stats.grok.se/. 2014-04-22.

[52] Laura R. Moore, Kathryn Bean, and Tariq Ellahi. Transforming reactive auto-scaling into proactive auto-scaling. In *Proceedings of the 3rd International Workshop on Cloud Data and Platforms*, CloudDP '13, pages 7–12, New York, NY, USA, 2013. ACM.

[53] Robert Nau. Forecasting course. http://people.duke.edu/ rnau/compare.htm. 2014-04-25.

[54] Hiep Nguyen, Zhiming Shen, Xiaohui Gu, Sethuraman Subbiah, and John Wilkes. Agile: Elastic distributed resource scaling for infrastructure-as-a-service. In *Presented as part of the 10th International Conference on Autonomic Computing*, pages 69–82, Berkeley, CA, 2013. USENIX.

[55] Karlsruhe Institute of Technology. Limbo. http://sdqweb.ipd.kit.edu/mediawiki-descartes/index.php/Tools. 2014-04-08.

[56] University of Toronto. Snowflock. http://sysweb.cs.toronto.edu/snowflock. 2014-03-28.

[57] Oracle. Java. http://www.oracle.com/technetwork/java/index.html. 2014-03-30.

[58] Christy Pettey. Gartner symposium/itxpo 2007. http://www.gartner.com/newsroom/id/503867. 2014-02-09.

[59] Martin Arlitt Raoufehsadat Hashemian, Diwakar Krishnamurthy. Web workload generation challenges - an empirical investigation, 2010.

[60] RightScale. Rightscale. http://www.rightscale.com/. 2014-03-26.

[61] George Athanasopoulos Rob J Hyndman. Forecasting: principles and practice. https://www.otexts.org/fpp/. 2014-02-09.

[62] Slava Razbash Drew Schmidt Zhenyu Zhou Yousaf Khan Christoph Bergmeir Earo Wang Rob J Hyndman, George Athanasopoulos. Forecast package for r. http://robjhyndman.com/software/forecast/. 2014-03-14.

[63] Robert Gentleman Ross Ihaka. R project. http://www.r-project.org/. 2014-03-30.

[64] N. Roy, A. Dubey, and A. Gokhale. Efficient autoscaling in the cloud using predictive models for workload forecasting. In *Cloud Computing (CLOUD), 2011 IEEE International Conference on*, pages 500–507, July 2011.

[65] Ilari Shafer, Kai Ren, Vishnu Naresh Boddeti, Yoshihisa Abe, Gregory R. Ganger, and Christos Faloutsos. Rainmon: An integrated approach to mining bursty timeseries monitoring data. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '12, pages 1158–1166, New York, NY, USA, 2012. ACM.

[66] Zhiming Shen, Sethuraman Subbiah, Xiaohui Gu, and John Wilkes. Cloudscale: Elastic resource scaling for multi-tenant cloud systems. In *Proceedings of the 2Nd ACM Symposium on Cloud Computing*, SOCC '11, pages 5:1–5:14, New York, NY, USA, 2011. ACM.

[67] Star's. Network statistics from star's. http://www.ispreview.co.uk/story/2010/06/15/world-cup-breaks-global-broadband-internet-traffic-record-uk-isps-unharmed.html. 2014-04-22.

[68] Statsoft. Time-series-analysis. http://www.statsoft.com/Textbook/Time-Series-Analysis. 2014-02-08.

[69] James Taylor. Short-term electricity demand forecasting using double seasonal exponential smoothing. *Journal of the Operational Research Society*, 54(8):799–805, August 2003.

[70] Rice University. Rubis. http://rubis.ow2.org/. 2014-03-08.

[71] Simon Urbanek. Java-r interface (jri). http://rforge.net/JRI/. 2014-03-30.

[72] Dr.Padmavathi Ganapathi V V.Vinothina Dr.R.Sridaran. A Survey on Resource Allocation Strategies in Cloud Computing. *International Journal of Advanced Computer Science and Applications(IJACSA)*, 3, 2012.

[73] Jóakim Gunnarson von Kistowski, Nikolas Roman Herbst, and Samuel Kounev. Modeling Variations in Load Intensity over Time. In *Proceedings of the 3rd International Workshop on Large-Scale Testing (LT 2014), co-located with the 5th ACM/SPEC International Conference on Performance Engineering (ICPE 2014)*, pages 1–4, New York, NY, USA, March 2014. ACM.

[74] Ben Wun. Survey of software monitoring and profiling tools. http://www.cse.wustl.edu/ jain/cse567-06/ftp/sw[underscore]monitors2/index.html. 2014-04-13.

[75] Rerngvit Yanggratoke, Fetahi Wuhib, and Rolf Stadler. Gossip-based resource allocation for green computing in large clouds. In *Proceedings of the 7th International Conference on Network and Services Management*, CNSM '11, pages 171–179, Laxenburg, Austria, Austria, 2011. International Federation for Information Processing.